CONCEPTUALISATION, DEVELOPMENT AND EVALUATION OF A NOVEL FRAMEWORK TO ENHANCE DATA SECURITY IN MOBILE CLOUD COMPUTING ENVIRONMENT

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY, AUCKLAND, NEW ZEALAND.
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisors

Dr Krassie Petrova

Dr Mee Loong (Bobby) Yang

Professor Stephen MacDonell

July 2022

By

NOAH OGHENEFEGO OGWARA

School of Engineering, Computer and Mathematical Sciences

LIST OF PUBLICATIONS

These are the publications resulting from this study

1. **Ogwara, N. O**., Petrova, K., & Yang, M. L. (2019). Data security frameworks for mobile cloud computing: A comprehensive review of the literature. In *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)* (pp. 1-4). IEEE.
   DOI: 10.1109/ITNAC46935.2019.9078007

2. **Ogwara, N.O**, Petrova, K., Yang, M. L. B & MacDonell, S. G. (2020). Enhancing Data Security in the User Layer of Mobile Cloud Computing Environment: A Novel Approach. In *2020 Security and Management (SAM2020) Computational Science and Computational Intelligence (CSCI)* (pp. 129-146). Springer.
   https://ui.adsabs.harvard.edu/abs/2020arXiv201208042O/abstract

3. **Ogwara, N. O**., Petrova, K., & Yang, M. L. B. (2020). MOBDroid: an intelligent malware detection system for improved data security in mobile cloud computing environments. In *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)* (pp. 1-6). IEEE.
   DOI:10.1109/ITNAC50341.2020.9315052

4. **Ogwara, N. O**., Petrova, K., Yang, M. L. B., & MacDonell, S. (2021). Enhancing Data Security in the User Layer of Mobile Cloud Computing Environment: A Novel Approach. *Advances in Security, Networks, and Internet of Things*, 129-145.
   https://link.springer.com/chapter/10.1007/978-3-030-71017-0_10

5. **Ogwara, N. O**., Petrova, K., & Yang, M. L. (2021). MOBDroid2: An Improved Feature Selection Method for Detecting Malicious Applications in a Mobile Cloud Computing Environment. In 2021 Conference Proceeding of Computational Science and Computational Intelligence (CSCI). IEEE.
   DOI:10.1109/CSCI54926.2021.00137

6. **Ogwara, N. O**., Petrova, K., & Yang, M. L. (2022). Towards the Development of a Cloud Computing Intrusion Detection Framework Using an Ensemble Hybrid Feature Selection Approach. *Journal of Computer Networks and Communications*, *2022*.
   DOI: https://doi.org/10.1155/2022/5988567

7. **Ogwara, N. O**., Yang, M. L. & Petrova, K., (2022) Towards the Development of an Application Risk Assessment Model using Ensembling Technique for Improved Security in Mobile Cloud Computing Environment. In Progress.

8. **Ogwara, N. O**., Petrova, K., & Yang, M. L. (2022) MINDPRES: A Prototype System to Enhance Data Security in the User Layer of Mobile Cloud Computing Environment. In Progress

ACKNOWLEDGEMENTS

I want to use this medium to express my sincere gratitude to my primary supervisor, Dr. Krassie Petrova, for her immense support and constant feedback during this research work. Without her efforts and guidance, this research work would not have been completed.

I am also grateful to my secondary supervisor, Dr Bobby Mee Loong Yang, for his support and feedback for this study.

I sincerely thank Professor Steve MacDonell for his mentorship role that made this research work a great success. His constant feedback and directions have helped improve the quality of this research work.

My appreciation goes to the Auckland University of Technology (AUT), New Zealand Scholarship office staffs and School of Engineering, Computer and Mathematical Science AUT for their financial support for my doctoral scholarship that has made my journey to AUT from my home country a great success. I want to appreciate all members of AUT Network Security Research Group (NSRG) for their feedback before my PGR9 presentation that has help me to improve the quality of this study.

Furthermore, I thank God almighty for his grace and mercy upon my life, which enabled me to complete this research work.

Finally, my sincere appreciation goes to my wife, Mrs Violetina Ehimuan Ogwara and my son, Ethan Oghenefego Noel Ogwara for their moral support for this study. I want to thank my parents (Chief and Mrs Jonathan Ogwara) and my siblings (Elijah Ogwara, Eloho Ogwara and Faith Ogwara). I want to thank Professor One-time Obi Obeten Ekabua, Dr Friday Okwonu and Dr Nicholas Oluwole Ogini for their support, especially when I started the PhD admission process. I also want to use this opportunity to thank my friends for all their support during my PhD journey (Professor Moses Okechukwu Onyesolu, Dr Franklin Okorodudu, Mrs Augusta Aghaulor, Dr Eghbal Ghazizadeh, Confidence Wanogho, Gbenga Wahab Adeyemi, Pastor Martins Ayo, Pastor Busola Martins, Ojabo Myles, Trust Okoroego, Temitope Oguntade, Anthony Ezeamaka, Obichukwu Onyeowuzoni, Obi Azuibike, Belinda Idowu, Hariata Emeka, Anthony Okwoani, Clement Chima, and Ekpemuaka Charles).

TABLE OF CONTENTS

List of Publications……ii
Acknowledgements……iii
Table of Contents……iv
List of Figures……viii
List of Tables……x
Declaration……xii
Abstract……xiii

TABLE OF CONTENTS iv
CHAPTER ONE 1
INTRODUCTION 1
1.1 Background of the Study 1
1.2 Problem Statement 4
1.3 Research Goal and Objectives 7
1.4 Thesis Organization 8
1.5 Chapter Summary 9
CHAPTER TWO 10
LITERATURE REVIEW 10
2.1 Overview of Mobile Cloud Computing 10
2.1.1 Mobile Cloud Computing Service Models 15
2.1.2 Mobile Cloud Computing Challenges 15
2.2 Security and Privacy Issues in Mobile Cloud Computing 16
2.2.1 Data Security Issues 16
2.2.2 Partitioning and Offloading Security Issues 17
2.2.3 Virtualization Security Issues 17
2.2.4 Mobile Cloud Applications Security Issues 18
2.2.5 Mobile Device Security Issues 18
2.2.6 Privacy Issues 19
2.3 Mobile Cloud Computing Security Vulnerabilities 20
2.3.1 Vulnerabilities in Application and Interface Layer 20
2.3.2 Vulnerabilities in Platform Layer 21
2.3.3 Vulnerabilities in Infrastructure Layer 22
2.4 Mobile Cloud Computing Security Requirements and Threats 23
2.4.1 MCC Threat Trend Analysis 27
2.4.2 The Egregious CC Threat Analysis Mapping Using the STRIDE Model 27
2.5 Analysis of Current MCC Data Security Frameworks 29
2.5.1 Review of the MCC Data Security Frameworks 31
2.5.2 Summary of the Review 40
2.6 Intrusion Detection System 42
2.6.1 Intrusion in The Mobile Devices 43

2.6.2 Intrusion in The Cloud-Infrastructure 45
2.6.3 Types of Intrusion Detection System 46
2.6.4 Intrusion Detection Methods 48
2.7 Analysis of Current IDS Frameworks in CC, MD and MCC Environment 49
2.7.1 Review of IDS Frameworks that Targets CC Infrastructure 51
2.7.2 Review of IDS Frameworks that Targets MD Infrastructure 52
2.7.3 Review of IDS Frameworks that Targets MCC Infrastructure 53
2.7.4 Summary of the Review 53
2.8 State of Security for Mobile Devices in The MCC Environment 57
2.9 Mobile Application Risk Factors and Assessment 60
2.10 Research Gaps 62
2.11 Chapter Summary 64
CHAPTER THREE 65
DESIGN SCIENCE RESEARCH METHODOLOGY 65
3.1 Research Methodology 65
3.2 Problem Identification Phase 68
3.3 Solution Design Phase 69
3.3.1 The Proposed MCC Data Security Framework 69
3.3.2 The Proposed Prototype System 71
3.4 Evaluation and Communication Phase 73
3.5 Chapter Summary 74
CHAPTER FOUR 75
DATA COLLECTION, ANALYSIS AND LABORATORY EXPERIMENTS 75
4.1 Android Operating System Security Description 75
4.2 Data Collection 78
4.2.1 Dataset Construction 81
4.2.2 Permission and Intent Usage Analysis 84
4.3 Laboratory Experiment 1 90
4.3.1 Machine Learning Classification Algorithms 91
4.3.2 Validation Metrics used in This Study 94
4.3.3 Evaluation Metrics used in This Study 94
4.3.4 Results Obtained from Experiment 1 95
4.3.5 Discussions of results Obtained from Experiment 1 97
4.4 Laboratory Experiment 2 99
4.4.1 Feature Selection Methods 100
4.4.2 The Proposed Filter-Based Feature Selection Technique 100
4.4.3 The Proposed Ensemble ML Model Using Static Analysis Approach 104
4.4.4 RESULTS Obtained from Experiment 2 105
4.4.5 DISCUSSIONS of results Obtained from Experiment 2 106

4.5 Laboratory Experiment 3 109
4.5.1 App Dynamic Features Extraction and Dataset Construction 110
4.5.2 The Proposed Ensemble ML Model Using Dynamic Analysis Approach 111
4.5.3 Discussions of results Obtained from Experiment 3 112
4.6 Chapter Summary 112
CHAPTER FIVE 113
PROTOTYPE DESIGN AND IMPLEMENTATION 113
5.1 Prototype Design 113
5.1.1 The Device Manager 114
5.1.2 The App Evaluator 115
5.1.3 The Detection Engine 121
5.2 Prototype Implementation 124
5.2.1 The Database Design 124
5.2.2 The Implementation Tools 126
5.2.3 The Unified Modelling Of The Prototype System 131
5.2.4 The Algorithmic Design Of The Prototype System 135
5.3 Prototype Testing 137
5.3.1 Test Case Scenario 1 139
5.3.2 Test Case Scenario 2 140
5.3.3 Test Case Scenario 3 141
5.3.4 Test Case Scenario 4 142
5.3.5 Test Case Scenario 5 143
5.4 Chapter Summary 144
CHAPTER SIX 145
PROTOTYPE EVALUATION 145
6.1 Prototype Evaluation 145
6.1.1 Experimental Setup For The Prototype Performance Evaluation 146
6.1.2 Description Of The Evaluation Testbed 146
6.1.3 Phase One (The App Evaluator) Results 148
6.1.4 Phase Two (The Detection Engine) Results 151
6.1.5 Performance Evaluation Results 152
6.1.6 Energy Consumption Evaluation Of The Prototype System 154
6.2 Prototype System Expert Evaluation Feedback 156
6.2.1 Expert 1 Feedback 156
6.2.2 Expert 2 Feedback 156
6.2.3 Expert 3 Feedback 158
6.2.4 Expert Feedback Summary 158
6.3 Results Comparison with Related Works 159
6.4 Chapter Summary 162

CHAPTER SEVEN 163
DISCUSSION AND CONCLUDING REMARKS 163
7.1 Overview of the Study 163
7.2 Addressing the Research Questions 164
7.3 Research Contribution 167
7.4 Challenges and Limitations of the Study 168
7.5 Directions for Further Research 169
References 171
APPENDIX A (TABLES) 187
APPENDIX B (UML DIAGRAMS) 199
APPENDIX C (SOURCE CODES) 203

LIST OF FIGURES

Figure 1.1 The General Architecture of MCC.....3
Figure 1.2 Mobile Cloud Architecture 4
Figure 2.1 CC Threat Ranking Change Spectrum.....27
Figure 2.2. Summary of the Address Threats by Existing Frameworks.....41
Figure 2.3. Summary of the Address Security Requirements by Existing Frameworks.....42
Figure 2.4. Analysis of IDS Frameworks and their Target Environment.....55
Figure 2.5. Analysis of IDS Frameworks for ML & IPS Components.....55
Figure 3.1: Research Process Followed in this Study.....67
Figure 3.2: The Proposed MCC Data Security Framework.....71
Figure 3.3 The Proposed Prototype System (MINDPRES).....72
Figure 4.1 Data Pre-processing and Features Extraction Stages.....82
Figure 4.2 Top 25 Permission Usage Frequency of Apps in the Dataset.....87
Figure 4.3 Top 25 Intent Usage Frequency of Apps in the Dataset.....88
Figure 4.4 Performance Evaluation Results for the Ten ML Classifiers (CA,PR,RC & FM).....98
Figure 4.5 Performance Evaluation Results for the Ten ML Classifiers (ER,FPR,FNR, & FAR).....99
Figure 4.6 The Proposed Ensemble ML Model using Static Analysis Approach.....105
Figure 4.7 Ensemble ML Model Performance Results Using the FS Approach.....107
Figure 4.8 Ten-Fold Cross Validation Results of the ML Training.....108
Figure 4.9 Ensemble ML Model Training and Testing Time.....108
Figure 4.10 The Proposed Ensemble ML Model Using Dynamic Features Analysis Approach.....111
Figure 5.1 High-Level View of the Proposed Prototype System Implementation (MINDPRES).....114
Figure 5.2 The Device Manager.....115
Figure 5.3 The App Evaluator.....117
Figure 5.4 App Intrusion Manager.....122
Figure 5.5 MINDPRES-Database Entity Relationship Diagram.....124
Figure 5.6 MINDPRES-Device Manager UI.....127
Figure 5.7 MINDPRES-App Evaluator UI.....128
Figure 5.8 MINDPRES-App Detection Engine-Online Activities Tab UI.....129
Figure 5.9 MINDPRES-App Detection Engine-Malicious Activities Tab UI.....130
Figure 6.1 Risk Assessment Result of all Benign apps used in the Evaluation.....150
Figure 6.2 Risk Assessment Result of all Malicious apps used in the Evaluation.....151
Figure 6.3 Detection Performance Evaluation Result 154

Figure 6.4 Energy Consumption Evaluation Result 155
Figure 1. Class Diagram of the Prototype System.....199
Figure 2. Sequence Diagram of the Prototype System.....200
Figure 3. Activity Diagram of the Prototype System.....201
Figure 4. Component Diagram of the Prototype System.....201
Figure 5. Deployment Diagram of the Prototype System.....202
Figure 6. Energy Profiler Estimation Result of the Prototype System.....202

LIST OF TABLES

Table 2.1. The Seven Basic MCC Security Requirements.....24
Table 2.2. The evolution of the CC security threats.....25
Table 2.3. The Egregious eleven CC Threats.....26
Table 2.4 STRIDE Threat Analysis using the Egregious CC Threats.....28
Table 2.5 Analysis Summary of the Selected Data Security Frameworks.....30
Table 2.6. Analysis of IDS Frameworks 50
Table 2.7. Dimensions Adopted in this Study for Framework Analysis.....56
Table 2.8 Existing Validation Metrics Used for the Evaluation of Existing IDS and IDPS.....57
Table 4.1 Apps Apk Source Market Distribution.....81
Table 4.2 Sample Structure of the Constructed Dataset 1(Permissions).....83
Table 4.3 Sample Structure of the Constructed Dataset 2(Intents).....83
Table 4.4 Sample Structure of the Constructed Dataset 3 (Permissions and Intents).....84
Table 4.5 Permissions Usage Analysis.....85
Table 4.6 Intents Usage Analysis.....86
Table 4.7 The Selected ML Classification Algorithms.....91
Table 4.8 Apps Distribution as used in the three datasets.....91
Table 4.9 Results of the First Experiment.....96
Table 4.10 Ten-Fold Cross Validation Results of the Ten ML Classifiers.....96
Table 4.11 The Selected 39 Features from the Proposed Filter-Based FS method.....104
Table 4.12 Results of the Second Experiment.....106
Table 4.13 Network Traffic Apps Data Distribution.....109
Table 4.14 API Calls Features from the App Network Traffic Data.....110
Table 5.1 App Information Table.....124
Table 5.2 App Evaluation Table.....125
Table 5.3 App Network Traffic Table.....125
Table 5.4 App Online Activities Table.....125
Table 5.5 App Traffic Detection Table.....126
Table 5.6 Selected Permissions and Intent used in the design of the ensemble ML model.....138
Table 5.7 Risk Score Output (Dangerous Permissions) of the Statistical Model.....138
Table 5.8 Classification Outcome of the App Evaluator.....139
Table 5.9 Permission Requested by the Test App for Scenario 1.....139
Table 5.10 Permission Requested by the Test App for Scenario 2.....140
Table 5.11 Permission Requested by the Test App for Scenario 3.....141
Table 5.12 Permission Requested by the Test App for Scenario 4.....142
Table 6.1 Apps Distribution Sample Installed in Each Device.....146

Table 6.2 Benign Apps Downloaded from Google Play Store……...148
Table 6.3 Malicious Apps Downloaded from CICMalDroid2020……...148
Table 6.4 Prototype System Risk Assessment Evaluation Result using Both Benign and Malicious Apps.....149
Table 6.5 Network Activities of all Apps in the Device Captured by the Detection Engine.....152
Table 6.6 Detection Performance Evaluation Results of the Prototype System……153
Table 6.7 Energy Consumption of each Device.....154
Table 6.8 Results Comparison with Related Works.....161
Table 1 Unique List of Permissions Usage in the Constructed Dataset……...187
Table 2 Unique List of Intent Usage in the Constructed Dataset.....191
Table 3 Feature Selection Results of the Proposed Filter-Based FS Method……...195
Table 4 Related Works and their ML Classifiers Used……....198

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Noah Oghenefego Ogwara

ABSTRACT

Security issues (e.g., data breaches, malicious applications, account hijacking, and insecure application programming interfaces) are obstacles in the adoption of Cloud Computing (CC) and Mobile Cloud Computing (MCC) technologies as the enormous data circulation through the Internet has attracted attackers to this environment. MCC inherits the security challenges faced by CC that affect the security and privacy of user information, such as multi-tenancy, data security, virtualization security, and application vulnerabilities. The highly distributed nature of MCC makes it vulnerable to attacks such as denial of service (DoS), distributed denial of services (DDoS), virtual machines to virtual machines attacks, man-in-the-middle attacks, cloud malware injection, covert channel, and others. These attacks spread to the Mobile Device (MD) layer of the MCC infrastructure, in which external access to MDs may enable the stealing of sensitive information. The exposure of the MD is due in part to vulnerabilities introduced by malicious applications downloaded by users from trusted, or untrusted sources. Despite the significant attack exposure level of the MD layer of the MCC architecture, it has received little research attention; most of the existing work reported in the extant literature targets the cloud infrastructure of the MCC environment. Although some researchers have offered security solutions for the MCC environment, these solutions are not comprehensive enough as they only provide countermeasures to a small number of known security threats. Hence the main research question: what security components are required in a framework that can be used to protect MCC resources against attacks and enhance the security of user data in this environment? To address the main research question, this study adopted a Design Science Research Methodology (DSRM) approach, to identify the security components needed and proposes a novel security framework that offers a comprehensive solution to a large number of the known security threats in the MCC domain. Based on the framework, a proof-of-concept prototype system, a novel hybrid intrusion detection and prevention system named MINDPRES (Mobile-Cloud Intrusion Detection and Prevention System) was designed and implemented. MINDPRES aims to protect the security of the MD layer of the MCC infrastructure, it combines a host-based Intrusion Detection System (IDS) and a network-based IDS using a Machine Learning (ML) model for the detection of malicious activities at the MD nodes of the MCC environment. Android apk files from two repositories were collected and used to construct the datasets used in building an ensemble ML classification model that uses the permissions and intents demanded by apps to determine if an app is malicious or not. Using the prototype system (MINDPRES), MD users can evaluate all apps on their device; each app is assigned a risk score and risk category. The system also monitors the actual behaviour of the apps by analysing the API calls to detect malicious behaviour; the MD user is automatically alerted, and the activities of such apps are blocked. The results obtained from the experiments carried out in this study show that the prototype system is effective in tackling security issues caused by malicious apps in the user layer of the MCC environment. The energy consumption and intrusion detection performance evaluation results indicate that the prototype system is feasible for implementation in the resource-constrained MDs used in the MCC environment. In addition, the prototype system was evaluated by invited security experts who were given access to standalone MDs with MINDPRES pre-installed. The expert feedback was also positive, and they all agreed that the prototype system is highly effective in detecting and preventing malicious activities at the MD node of the MCC infrastructure. Despite the prototype implementation being limited to the Android mobile ecosystem, this study proposes a novel data security framework that detect and prevent the security issues caused by malicious applications in the MCC environment, by monitoring device behaviour using a hybrid analysis approach without root level access to the device resources. However, there is a need for further research to improve the proposed framework to manage security issues at other layers of the MCC architecture and the implementation of a cross-platform prototype system.

CHAPTER ONE
INTRODUCTION

This research identifies the security components that is required to tackle security issues faced by users of Mobile Cloud Computing (MCC) technology and proposes a novel data security framework that offers a better solution to known security threats in the MCC environment. Based on the proposed novel framework, this study implements a prototype system named **MINDPRES** (Mobile-Cloud Intrusion Detection and Prevention System) for the security of the User Layer (UL) of the MCC infrastructure as a proof of concept.

This chapter is organized as follows: A brief background of the study is presented in section 1.1; section 1.2 discusses the problem statement. The research goal, research questions and the objectives are presented in section 1.3. The overall thesis structure and the chapter conclusion are discussed in sections 1.4 and 1.5. respectively.

1.1 BACKGROUND OF THE STUDY

The evolution of Cloud Computing (CC) has witnessed rapid growth in the last decade. This rapid growth of the CC technology is a result of the invention of modern technology that depends on its infrastructure, for example, MCC, Internet of Things (IoT), Fifth and Sixth generation (5G and 6G) networks, and Software Defined Networks (SDN) (Kumar & Goyal, 2019).

CC can be described as computing resources and Information Technology (IT) services, made available on-demand through internet technology, in a pay-as-you-go business model (Hazarika et al., 2014). Cloud Services Providers (CSP) offer various cloud services to enterprises that function as customers service. CC provides services and computing resources to its end-user through a highly computational network with remote servers managed by the CSP. CC offers a model for various end-users to use software applications(apps), storage, and processing capabilities without investing in infrastructure. CC allows its end-users to use infrastructure (for example, networks, servers, and storage), platforms (for example, operating systems and middleware services), and software example application programs) provided by CSP, for example, Amazon, Google, Microsoft, and Salesforce at an affordable rate. CC user may regularly update the data stored in the cloud. However, data in this environment may be vulnerable to unauthorized modification, disclosure, and replay attacks during data transmission and storage in the cloud infrastructure (Dinh et al., 2013).

Data security during updates is essential to ensure storage accuracy in such a dynamic environment. To this end, it is necessary to ensure secure and reliable data transmissions between the cloud user and the cloud storage. CC has received significant attention from different researchers, and there is one important question that attracts serious attention. How do we complement the anytime access possibilities of the cloud with anywhere access? The answer to this question has focused on different research studies that proposed the Mobile Device (MD) as the cloud services consumption node.

The MD complements the CC infrastructure by offering access to enterprise information anywhere, anytime. However, a significant revolution that Mohiuddin et al. (2012) describe is "24x7x365 mobile access" since most MDs have reliable support for various networks connectivity between the cloud clusters and MDs via 3G, 4G, Wi-Fi, and Bluetooth (Meads et al. 2009).

Mobile Computing (MC) is one of the leading business solutions in the IT industry. MDs are increasingly becoming the most efficient and convenient communication tool in human life, not confined by time and place. The number of mobile users is increasing due to continually improving these devices' user-friendly hardware and software (Ba et al., 2013; Chung et al., 2014). Dinh et al. (2013) stated that MD users acquire rich experience of various mobile apps services that run on the devices or remote servers via wireless communication networks. MC has failed to satisfy the high computational resources required by MD users for data processing. Also, the limited resources of MD significantly hinder the improvement of its service qualities. However, MCC solves the limited resources problem faced by MD user by offloading high computational tasks and storage to the CC infrastructure (Noor et al.,2018).

MCC has attracted people's attention in business as a technology that lessens mobile services and applications running and development costs. MCC is the interconnection of CC, MC and wireless networks that provides powerful computational resources to MD users. Buyya et al.,2009; Mollah et al., 2012 stated that MCC is offered as a service of CC, which is used in either the mobile embedded environment or mobile phone environment. MC is well integrated with CC because of the vital attributes of the cloud model, such as on-demand self-service, resource pooling, measured services, rapid elasticity, and broad network access. MCC is a new model for mobile applications, in which data processing and storage are transferred from the device to robust and centralized computing platforms positioned in cloud. These applications are then accessed over the wireless network connection based on a thin local client or web browser (on the MD). MCC takes full advantage of CC infrastructure's availability, enabling MD users to offload computationally rigorously and high storage demanding tasks on available cloud resources using the wireless network. MCC as technology focuses on storing and processing data outside of the devices using the CC infrastructure. However, security and privacy remain critical issues of this technology.

Sanaei et al. (2012) describe MCC as a valuable MC technology that leverages the centralized elastic resources of various clouds and network technologies towards complete functionality, storage, and mobility which serves a plenitude of MDs anywhere, anytime through the internet, regardless of various environments and platforms based on the pay-as-you-use policy. The general MCC architecture is shown in Figure 1.1. The architecture provides an access point at which MDs can request services in the cloud through the mobile internet. These services are delivered via the communication channel to the MD user. In this regard, the middleware allows for cross platforms of

different MDs to manage the service request and ensures easy access to other MDs users' cloud resources in the MCC environment.

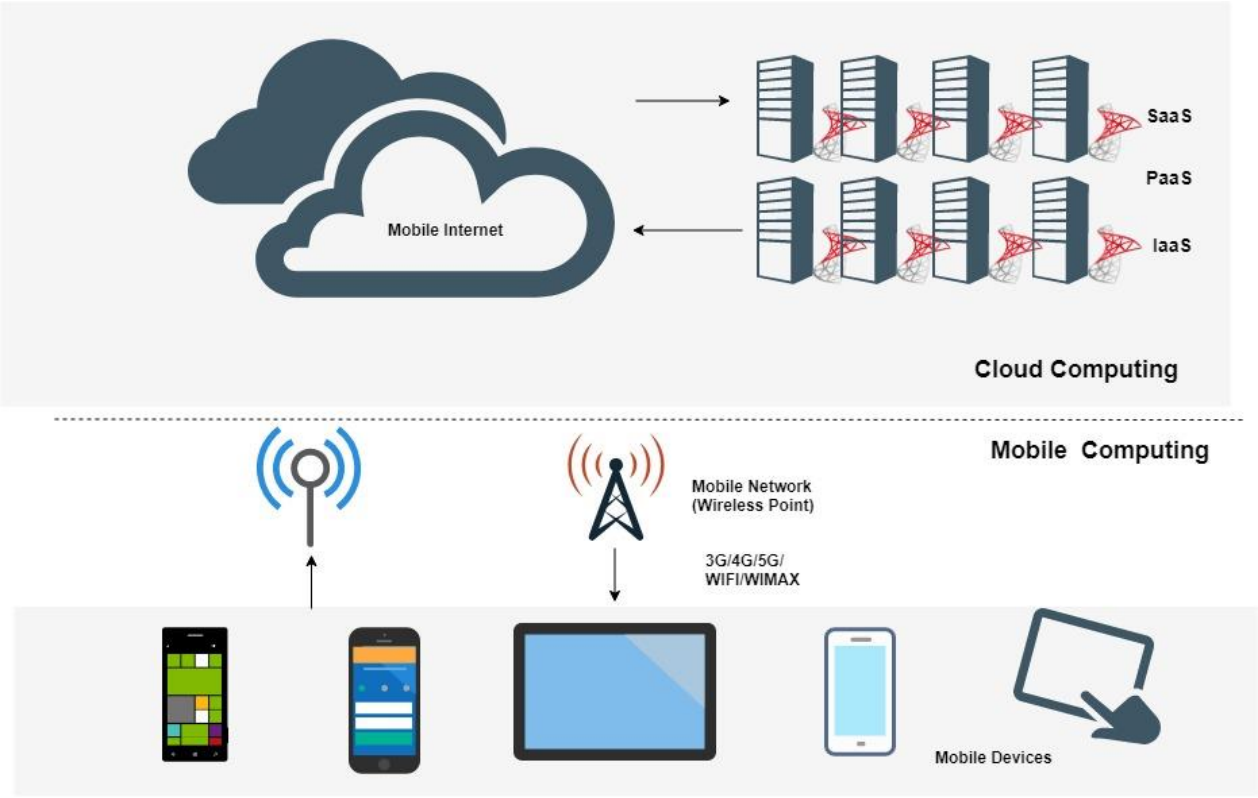

Figure 1.1 The General Architecture of MCC adapted from Dai et al. (2012)

The development of apps that runs on the MD platforms in the MCC environment has increased in recent times. Mobile application categories such as social media, apps using location-based services, games, and email apps have also contributed to the wide user acceptance and adoption. However, MD resource constraints, for example, device processor, energy consumption, and storage capacity, have posed some design challenges to the mobile apps' developer. MCC can be used to counter these challenges. Due to the new trend to use MCC technology for mobile apps development. MCC users can access the powerful cloud services anywhere, anytime on a pay as you use basis, even with the limited resources of their MDs. Using MCC can scale up or down rapidly to meet the MDs user demands as well as the MD capability (Noor et al., 2018).

Noor et al. (2018) describes the three different layers in MCC architectures. These layers include 1. Mobile Cloud User layer (MCUL) 2. Mobile Network Communication Layer (MNCL) and 3. Mobile Cloud Services Provider layer (MCSPL) as shown in Figure 1.2.

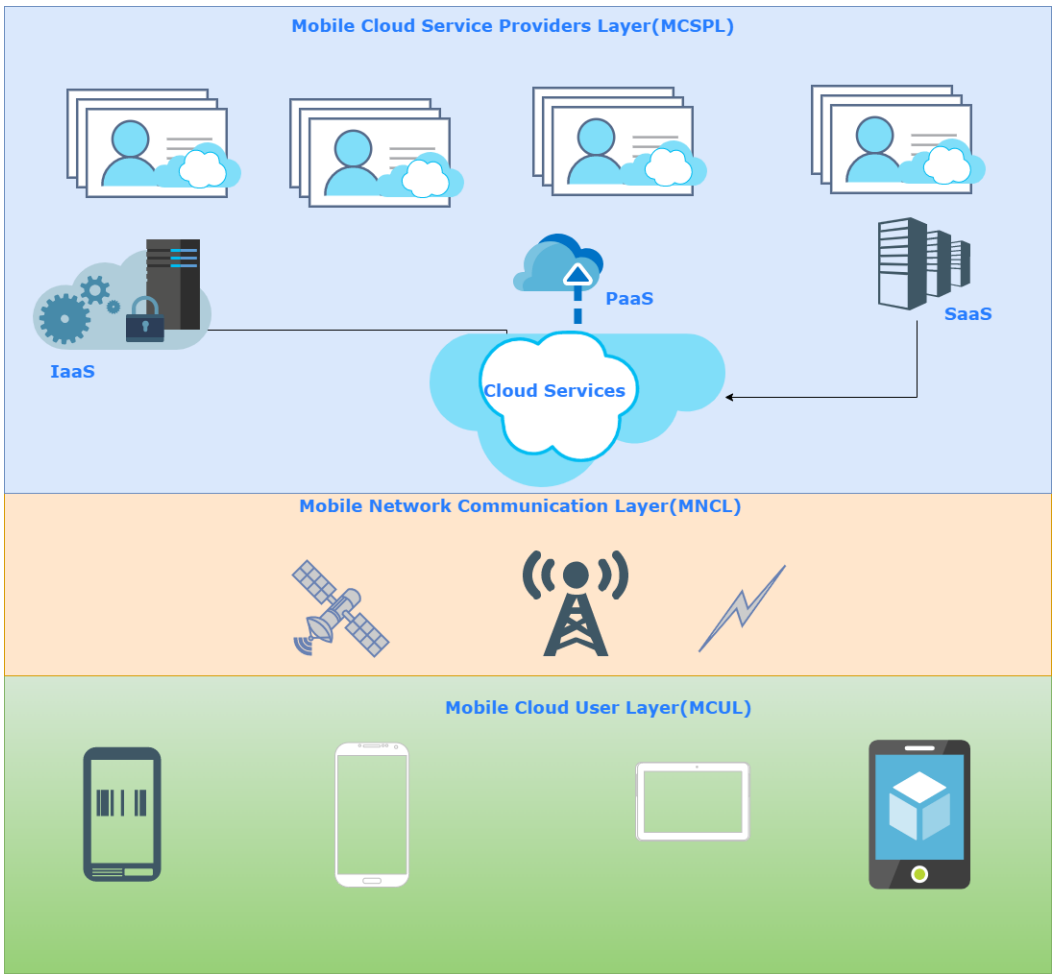

Figure 1.2 Mobile Cloud Architecture Adapted from Noor et al. (2018)

MCUL: This layer provides a platform in which users can gain access to the cloud resources using their MD through the MNCL.

MNCL: This layer is made up of different mobile networks that provide internet services to MD users request through their base stations to the mobile cloud.

MCSPL. This layer presents the MD user with the different CSP available, which they can use to store their information and deploy their cloud-based application.

1.2 Problem Statement

Security has become an obstacle in adopting CC and MCC technology even though the technologies can deliver a broad range of resources and services to its user (Sathye et al, 2022; Liang et al, 2021; Alghofaili et al, 2021; Tahirkheli et al,2021). The enormous data circulation through the internet has attracted attackers to this environment (Zkik et al., 2017). The mobile nature of the network nodes in MCC has raised numerous challenges related to data breaches, data loss, data replication, trust, security, and privacy (Zissis & Lekkas, 2012; Mollah et al., 2017, Dey et al., 2019). The security issues in MCC cut across the different models that make up its architecture (MCUL, MCNL and MCSPL). To protect this technology there is a need to develop a good security solution that provides

a strong defence against attacks on data which is beneficial to both the MCC users and the service provider.

Kulkarni & Khanai (2015) stated that even though users of MCC enjoy benefits such as device independence, reduced maintenance, reliability, reduced cost, and scalability, there is a need for MCC resources such as user data to be protected from all kinds of attacks. MCC inherits the security challenges facing CC, such as multi-tenancy, data security, virtualization security, which affect the security and privacy of user information stored in both mobile and cloud infrastructure. The distributed nature of MCC infrastructure allows intruders the possibility of gaining unauthorized access to these mobile and cloud resources, intending to extract sensitive information. However, the cloud infrastructure of the MCC environment is much more powerful and reliable than the mobile environment but the security and privacy of user data still poses a major challenge in the MCC environment (Mollah et al., 2017).

Furthermore, shared resources and virtualization features of the MCC environment have attracted cyber-attacks on the CC infrastructure such as a Denial of Services (DoS), Distributed Denial of Services (DDoS), virtual machines to virtual machine attacks, man-in-the-middle, cloud malware injection, covert channel, and so on. These attacks can spread to MDs and enable external access to MDs leading to the stealing of sensitive information of MD users (Inayat et al., 2017). Therefore, it is necessary to apply defensive measures to detect and prevent these attacks. In addition, the popularity of the Android mobile Operating System (OS) which currently holds over 85% of the market share of MD users in the MCC environment. These devices currently generate a large amount of traffic that exceed personal computers and made MDs a prime target for cyber-attacks. The use of MD in the MCC environment gives room for many security and privacy issues in this environment. However, protecting these devices is a challenging task due to rapid changes in technology which makes the MCC environment more complex (Gupta et al., 2018).

Nevertheless, the connectivity support for access points such as 2G/3G/4G/5G, Wi-Fi, Bluetooth and the open nature of the Android mobile OS exposes the MD to more sophisticated attacks that may affect the security requirements of the MCC environment. (Ribeiro et al.,2019). To this end, the 2019 Cloud Security Alliance (CSA) in 2019 mentions data breaches as the topmost threat in the CC domain. These raise concerns about user information security at the MD at the MCUL of the MCC environment.

Nisha et al. (2020) stated that MCC users enjoy unlimited computational resources. This technology enables them to execute resource-demanding applications. However, offloaded content execution is done on the cloud or edge servers in the MCC environment instead of the MD. The activities of malicious apps on the end-user device when offloading content for execution on the cloud or edge

server may affect the confidentiality and integrity of data stored in the cloud. The inability of the MD user to validate the correctness of the offloaded data at the point of usage is a critical concern. The integration of malicious codes into the cloud-based apps leads to attacks on the MCC infrastructure. These attacks may affect the integrity of both data and applications in the MCC environment (Mollah et al., 2017).

OS (2021) reported that malicious apps are amongst the most threatening security issues facing MCC users. This is because the malicious activities of such apps residing in this environment can affect both the MCUL and MCSPL of the MCC architecture. In addition, Mollah et al (2017) stated that malicious apps that reside on the user device and malicious insiders in the cloud network are major obstacles facing the security of user data in the MCC environment. The presence of malicious codes in MD during offloading affects the confidentiality and privacy of users in this domain. Kumar & Goyal (2019) affirmed that due to vulnerabilities in a software application that serves as the entry point to the cloud services using the internet affects the basic security requirements of the CC infrastructure which open doors to attacks. However, the conventional way of handling security attacks is not sufficient in the MCC environment. Therefore, it is necessary to develop a novel approach to handle threats causes by malicious apps in this domain and to provide a dependable MCC environment.

In recent times, attackers have adopted the use of malicious apps to target MD users in the MCC domain. These categories of malicious apps especially are hard to detect by existing defensive techniques, for example, malicious apps that have been modified by adding malicious payload to the original benign version of an app that has been already vetted by the app store, and can be successfully published in the app store when the signature of the benign app is compromised (Qi. et al.,2014; Idrees & Muttukrishnan,2014; Hou et al., 2016; Hatcher et al,2016; Ribeiro, et al.,2019; Zhou, et al., 2019).

In fact, traditional defensive mechanisms such as firewalls, access control, anti-spyware, anti-virus, and anti-malware, may not be strong enough to protect MDs in the MCC environment. Most of these defensive techniques require changing the kernel function of the mobile OS. Significant computational power is required for running the highly intensive computing security algorithms in the resource constrained MD. The need to shift these defensive mechanisms to the cloud end may have become necessary, as suggested by Inayat et al. (2017).

MCC has received extensive research effort conducted by academia’s and research organizations to provide a more secure MCC environment and attract more consumers to use these services. However, security and privacy have been reported by researchers as a major challenge that hinders this technology (Chen & Wang, 2011; Huang et al.,2011; Khan et al,2013; Goyal & Krishna, 2015;

Mollah et al., 2017; Khatri & Vadi, 2017; Chean et al.,2018; Noor et al;2018; Agrawal & Tapaswi, 2019; Dey et al., 2019).

This research work focuses on identifying the security components required in a framework that provides a better solution to tackle security threats faced by MCC user. The proposed solution in this study enhances the MCC environment's security and provides adequate protection to user data.

1.3 RESEARCH GOAL AND OBJECTIVES

The main goal of this study is to develop and evaluate a novel solution to the data security issues in the MCC environment. Its main Research Question (RQ) and Research Sub Questions (RSQ) can be formulated as:

Main RQ: What security components are required in a framework that can protect MCC resources against attacks and enhance the security of user data in the MCC environment?

RSQ1: Which specific MCC resources require to be protected to enhance the security of this environment?

RSQ2: What approach can be used to protect the identified MCC resources in RSQ1?

RSQ3: What metrics can be used to evaluate the performance of the identified approach in RSQ2 and how can this approach be implemented to protect the resource identified in RSQ1?

To address the main RQ, RSQs and meet the research goal, this study investigates the state-of-the-art MCC data security solution landscape, develop and evaluate a novel framework for MCC data security.

The following specific objectives will guide this study:

Research objective 1: To investigate the current MCC data security solution scope in-depth.

Research objective 2: To identify and analyze solutions that aim to detect, and protects MCC resources against, attacks on data.

Research objective 3: To propose, develop and evaluate a novel framework that enhances MCC data security in the user layer.

The main scope of this study covers the conceptualization of a novel framework to enhance data security issues in the MCC environment. The conceptualized enhanced framework includes different security technique across the three different architectural layers in the MCC environment. The study focuses on the development and Implementation of a prototype system that addresses security issues in the user layer of the MCC environment. The conceptualized enhanced framework implemented as a prototype system as a proof of concept in this study adopted the use of ensemble Machine Learning (ML) techniques with Intrusion Detection and Prevention System (IDPS) to detect

the security issues caused by malicious apps in the user layer of the MCC environment. Other security aspects such as digital forensic techniques are beyond the scope of this study and are reported as area of research for future studies.

1.4 THESIS ORGANIZATION

The thesis is organized as follows: Chapter two discusses the MCC security requirements and provides a threats analysis, followed by a detailed analysis of related works. The analysis of related works includes a comprehensive review of data security frameworks that propose security solutions in the MCC domain and a review of security frameworks that aims to detect and protect resources in CC, MC and MCC domains. The research gaps identified in the review of literature are presented at the end of chapter two.

Chapter three presents the research methodology adopted for this study and discusses how the methodology is applied to the body of research work conducted in this study. The proposed framework and a brief description of the proposed prototype system are also presented.

Chapter four discusses the methods involved in the processes of data collection and analysis, laboratory experiments of the ML Models, experimental evaluation and analysis of laboratory results are also presented.

Chapter five discusses the design and implementation of the prototype system as a proof of concept with respect to the proposed framework. The prototype system address security issues that face the MCUL of the MCC infrastructure. The testing of the prototype system are presented in chapter five with various test case scenarios.

Chapter six discusses the performance evaluation of the prototype system using real-life Android devices with popular Android mobile apps. The expert evaluation report presented in chapter six is based on the personal opinion of each invited security experts that participated in the evaluation of the prototype system implemented in this study.

Chapter seven conclude the thesis by discussing answers to the research questions and identifies the contribution of this research to the body of knowledge. Chapter seven also discussed the challenges, limitations of the study and possible research directions for future work.

1.5 CHAPTER SUMMARY

MCC inherits security challenges facing the CC, such as multi-tenancy, data security, virtualization security, and application vulnerabilities that affect the security and privacy of user information. The use of malicious apps targeting the MD of the MCUL in the MCC domain has raised security concerns that needs a better solution. The distributed nature of MCC has led to attacks in this environment such as Denial of Services (DoS), Distributed Denial of Services (DDoS), virtual machines to virtual machine attacks, man-in-the-middle attacks, cloud malware injection, covert channel attacks and others. When these attacks spread to the MCC infrastructure's user layer (MCUL), they may enable adversarial external access to MDs and the stealing of sensitive information from both device and cloud storage. This chapter discusses the research goal and the objectives that are required to address the main research problem. The next chapter provides a comprehensive review of the literature and identifies the research gaps addressed in this study.

CHAPTER TWO

LITERATURE REVIEW

The previous chapter provides insight into this work by discussing the background of the study, research problem, research goals and objectives, and how this thesis is organized.

This chapter presents background studies in MCC technology. The security challenges, security threats, and requirements of the MCC environment are also discussed. A detailed analysis of data security frameworks proposed in extant literature that adopted intrusion detection and prevention security mechanisms and policies in MC, CC, and MCC environments is presented. The state-of-the-art security in mobile devices is discussed alongside the research gaps identified in the entire literature.

2.1 Overview of Mobile Cloud Computing

The MCC paradigm combines several technologies, including CC, MC, and wireless networks (WN), to provide integrated services to users and organizations. MCC technology has grown in popularity over the years because of factors such as network mobility and dynamicity, MD independence, ubiquitous data access, and improved data communications (Moorthy et al, 2020). MCC entails the technological synchronization of the MDs' OS and the dynamic quality of cloud services. It is predicated on the fundamental characteristics of CC technology, for example, adaptability, elasticity, availability, scalability, and resource sharing (Dinh et al.,2013).

MC is contingent upon the ability of MDs to access computer resources. Additionally, MC facilitates the performance of tasks that were previously performed by traditional desktop computers. In general, three fundamental concepts underpin MC: hardware, software, and communication (Dinh et al., 2011; Liu et al., 2010). Hardware refers to user-accessible gadgets (for example tablet PCs and cell phones). The software includes applications that are designed and developed to perform tasks in a mobile environment, while communication encompasses networks and protocols that enable mobile computers to communicate, such as Wireless Local Area Networks (WLAN), Long-Term Evolution (4G LTE), and satellite networks.

The following are supported by the MC environment. First, there is mobility, which enables mobile nodes or fixed nodes to communicate with other devices' nodes via Mobile Support Stations (MSS) (for example servers and access points). Second, diversity of network access types refers to mobile nodes that can communicate via a variety of different access networks, such as Long-Term Evolution 4G LTE or Wireless Wide Area Network (WWAN), each of which has a unique communication bandwidth and overhead between the mobile nodes and the MSS.

Thirdly, frequent network disconnection indicates that mobile nodes are unable to maintain a continuous connection due to restricted resources such as battery energy and communication capacity.

Fourth, in terms of dependability and security, mobile node signals are susceptible to interference and eavesdropping in mobile networks, highlighting the growing importance of security in MC.

Furthermore, the MCC paradigm evolved to combine the advantages of MC and CC to efficiently utilize data centre computing capabilities and distribute them as mobile services. MC refers to devices with limited hardware, software, and communication capabilities, with mobility as the primary criterion. CC is a method of delivering enormous computing resources as services through virtualization and service-oriented techniques to cut costs, increase performance, and enable remote access. MCC enables the delivery of powerful computing resources as services. This enables low-resource mobile devices to do complicated computations that would otherwise require more powerful computer resources (Fernando et al., 2013).

MCC is a cloud-based computing system that enables resource-constrained MDs to run computationally heavy applications and store their data in the cloud. MCC has greatly improved execution speed and energy usage by shifting resource-intensive applications from hosting devices to cloud-based resources. The evolution of MCC reduces MDs' heavy computational requirements in data processing because all data and sophisticated computations are managed remotely via cloud-based resources. MCC has transformed the landscape of traditional MC over the last few years by enabling on-demand, self-service, measurable, elastic, and broad access mobile services. MCC devices such as smartphones and tablets, are becoming an increasingly vital component of our modern and virtual lifestyles (Noor et al.,2018).

Mobile application execution in the MC environment is computationally intensive, and as a result, MDs consume a significant amount of energy. The computational offloading technique was introduced to tackle the high demand for energy consumption by MDs in the MCC environment. In the MCC environment, computationally intensive applications and tasks are offloaded to the cloud for execution, and the results are returned to the MDs. Mollah et al. (2017) described the computational offloading steps in the MCC environment as partitioning, migration, and execution. However, the execution and processing tasks are shifted from the MDs environment to the cloud environment. The MDs retain control over how the tasks are executed and how much computation is offloaded to the cloud based on the available MCC resources.

With the rapid growth of MDs, developers are creating a plethora of applications for them, many of which offer cloud-based services with a rich user experience (Bahrami, 2015). These applications enable MCC users to access cloud-based rich experiences and services, even on low-resource MDs.

These applications must instantly scale up or down to meet the requirements of MCC users and the capabilities of mobile devices. To cloud-deploy a mobile application, it must first be segmented into components based on its requirements. Components of an application that rely on locally available mobile resources, such as various sensors, do not require cloud offloading. However, components that consume many resources must be executed in the cloud. As a result, these applications can be classified as client-side, client-cloud-side, or cloud-based. Most of the application's execution occurs on the mobile device in a client-based model. A client-cloud model, on the other hand, partitions an application into components that are executed on both a mobile device and a remote cloud. Whereas in a cloud-based model, the cloud is an integral part of the application, acting as the execution, processing, and storage location (Mollah et al, 2017).

Mobile virtualization is the most advanced feature emerging in today's world, and its applications for MDs are growing daily. The mobile user base continues to grow because it makes work easier and faster, provides cutting-edge technology, and enables users to access all apps via the network from anywhere in the world. Although MCC as a technology has a significant advantage in that it is extremely versatile, allowing us to access data and share information from anywhere in the world via the internet. It also offers cost-effectiveness, with usage and maintenance becoming relatively low, as well as real-time data availability, with all user information available in real-time on our MD when connected to the internet, from which we can update and share information.

For a variety of reasons (for example, mobility, communication, and portability), CC has long been recognized as a viable alternative to MC (Al_Janabi & Hussein, 2019). The following additions demonstrate how the cloud can be utilized to circumvent barriers in MC, emphasizing the benefits of MCC.

Extending battery lifetime: Battery life is a major problem when it comes to MDs. Numerous strategies have been proposed to improve CPU performance and to intelligently manage the disk and display to reduce power consumption. However, many solutions require structural changes to MDs or the inclusion of additional hardware, which adds to the cost and may not be practical for all MDs. This is also important form the perspective of security protection of the MD. In the MCC environment some of the security solution tasks required to protect the MDs can be offloaded to the cloud while some of its tasks that are not battery demanding can reside on the device to have real-time access to malicious activities that occur on the devices and offload the heavy security computational tasks to the cloud for further processing whenever a malicious activity is detected. These security tasks on the devices need to be running in the background and thus have high power demands to prevent battery exhaustion attacks suffered by most MD user.

Improving data storage capacity and processing power: MDs' storage capacity is also a constraint. MCC was created to enable mobile users to store and retrieve massive amounts of data

in the cloud using wireless networks. For instance, Amazon Simple Storage Service (Amazon S3) provides file storage capabilities. Another example is image exchange, which makes extensive use of cloud-based storage for mobile users. This is also important from the security perspective; as MD user shave no control - whenever their data are offloaded there is no way they can verify the integrity of their outsourced data. Although in recent times, users have developed trust based on the services provided by CSPs.

Increased reliability: Storing data or running applications in the cloud is an effective technique to increase reliability, as the data and apps are kept and backed up on a distributed network of computers. This significantly reduces the likelihood of data and application loss on mobile devices. Additionally, MCC may be used to create a comprehensive data security paradigm for service providers and end-users alike. For instance, the cloud can be utilized to safeguard protected digital content (e.g., videos, clips, and music) against exploitation and unauthorized distribution. However, the security of user data and applications is still questionable whenever there is security breach in the CSP area where there are possibilities of unauthorized distribution and exploitation.

Dynamic provisioning: Dynamic on-demand provisioning of resources on a fine-grained, self-service basis enables service providers and mobile users to operate apps without reserving resources in advance. This is also important from the perspective of security protection of the MD by constantly monitoring apps activities and reporting to the CSP of possible intrusion from a compromised MD node.

Scalability: Due to flexible resource provisioning, mobile apps may be deployed and scaled to meet unforeseen user demands. Providers of services can simply add and extend applications and services with little or no constraint on resource utilization. This is also important to the security of the MCC environment to help with the investigation of security breaches (including digital forensics) that occur, by identifying malicious nodes that are compromised in a distributed network. User activities can be reviewed by analysing the resource usage of different apps and devices within a particular period.

Multi-tenancy: Service providers (for example, network operators and data centre operators) can pool resources and prices to support a diverse set of applications and a large number of customers. The security impact of multi-tenancy on the user in the cloud environment includes data security, data loss and data theft. Access can be mistakenly given to an unauthorised individual by the database administrator. There are still security issues even though software and CC businesses claim that client data is safer than ever on their servers.

Integration Ease: Multiple services from disparate sources can be readily integrated via the cloud and the Internet to suit consumers' requests. This also important to protect the MD as security

vulnerabilities associated with different services integrated into the CC environment to suit user demands can also result in security threats faced by users in this environment.

Numerous mobile applications have incorporated some of MCC's benefits, such as mobile commerce, which is a business model for commerce conducted via mobile devices. Generally, mobile commerce applications perform certain functions that demand mobility (e.g., mobile transactions and payments, mobile messaging, and mobile ticketing). These apps confront a variety of obstacles (e.g., limited network speed, high complexity of mobile device setups, and security), which necessitate their integration into the MCC environment. Yang et al. (2010) presented a cloud-based 3G e-commerce platform. This paradigm combines the benefits of 3G networks and CC to boost the speed and security of data processing using PKI (public key infrastructure) (Dai & Zhou, 2010). To secure the privacy of the user's access to the outsourced data, the PKI mechanism employs encryption-based access control and over-encryption.

Apart from the business and commercial implications of MCC across diverse applications, platforms, international trade regulations, and transnational information and money flows, one of the most significant fundamental disadvantages of MCC, as currently understood, is the entire area of safe and trusted computing, which encompasses critical aspects of security, privacy, identity management, audit, and digital forensics (Al_Janabi, 2020). While numerous modern scholars are addressing a number of these subjects and difficulties, it is a daunting obstacle that must be conquered by more advanced design and development of new frameworks, architectures, secure open system protocols and processes that are globally standardized. Success in these areas can result in massive dividends and payoffs across several technical verticals and horizontal enterprise and commercial sectors in the MCC environment.

The last decade has seen numerous changes in our perception of computing and mobility. With the advent of CC and MCC, computing is increasingly being viewed as the fifth utility, alongside critical infrastructure utilities such as water, electricity, gas, and telecommunications, and it is already providing a basic level of computing service that is considered necessary to meet the general community's daily needs on a global scale and context. MCC is the most recent paradigm presented to realize this vision through the effective fusion of MC and CC, which has proven to be a viable solution for mobile computing for a variety of reasons (e.g., mobility, communication, portability, and availability).

2.1.1 MOBILE CLOUD COMPUTING SERVICE MODELS

MCC's cloud services for its users are based on the following service models.

A. ***Mobile Network as a Service (MNaaS):*** In this service model, service providers provide network infrastructure, allowing consumers to establish their networks, manage their traffic, and connect to servers. For instance, consider the OpenStack Networking Service (www.openstack.org).
B. ***Mobile Cloud Infrastructure as a Service (MIaaS):*** Providers of this service model provide cloud infrastructure and storage to mobile users. iCloud (www.apple.com/icloud/) and Google Drive (www.google.com/mobile/drive/) are two examples.
C. ***Mobile Data as a Service (MDaaS):*** In this service model, service providers provide database-related services to enable mobile users to manage their data, conduct transactions, and perform other data-related tasks. Oracle's mobile cloud data service (www.oracle.com/cloud/daas.html) and CloudDB are two such examples (Lei et al., 2015).
D. ***Mobile App as a Service (MAppaaS):*** Users can access, use, and execute cloud-based mobile applications via a wireless network from anywhere and at any time with this service model. For instance, take a look at the Google Play Store (www.play.google.com/store/apps).
E. ***Mobile Multimedia as a Service (MMaaS):*** Users can access and manage multimedia services such as watching movies or playing games via a wireless network equipped with powerful hardware, in (Zhu et al. (2011), the authors present an MMaaS service model.
F. **Mobile Community as a Service (MCaaS):** In this service model, mobile users can create and manage a mobile social network or community to provide social networking or community services to other users. The following illustrates this type of service model (Kovachev et al., 2010).

2.1.2 MOBILE CLOUD COMPUTING CHALLENGES

While MCC has several benefits for mobile users and CSPs, it also faces several challenges that make it more complicated than traditional CC. The following section discusses the challenges confronting MCC technology.

A. ***Limited Resources of Mobile Devices***: Although various aspects of mobile devices have improved, such as computational processing power, storage capacity, and battery life, there are still some limitations compared to a personal computer. As a result, running resource-intensive applications on mobile devices is inconvenient.
B. ***Heterogeneity***: MCC is characterized by a high reliance on heterogeneous wireless mediums, which creates a more challenging environment than conventional cloud computing. This affects how wireless communications are managed, the quality of communications, the response time of applications, the delivery of services, the mobility of mobile devices, and

security. Additionally, the heterogeneous environment created by diverse infrastructures, platforms, and application services creates interoperability and portability challenges in MCC.

C. ***Elasticity***: MCC services, like cloud computing services, must be elastic and scalable. When demand exceeds available resources, service providers must address the situation. The unavailability of resources and service interruptions create a problem for cloud services provided to privileged users.

D. ***Application Services Issues:*** Due to MDs' limited resources and high energy consumption, specific data and computationally intensive applications cannot be deployed on them. As a result, to utilize cloud computing services on mobile devices, most computational processing must occur in the cloud. In contrast, a small amount of computational processing occurs on mobile devices. In this case, mobile users will experience delays in processing and service provision.

E. ***Security, Privacy and Trust Challenges:*** The security, privacy, and trust issues that arise in an MCC environment are more volatile than in a traditional cloud computing environment. Additionally, the lack of computational processing capability necessary to execute complex algorithms makes it inconvenient to run computationally intensive anti-malware applications on MDs and personal computers.

2.2 Security and Privacy Issues in Mobile Cloud Computing

MCC uses various established and emerging technologies, including partitioning, offloading, virtualization, outsourced storage, and mobile-cloud-based applications. This section discusses security and privacy issues associated with the MCC environment.

2.2.1 Data Security Issues

The significant data security challenge arises from mobile users' data being stored and processed in clouds located at service providers' locations. Data loss, data breach, data recovery, data locality, and data privacy are all examples of data-related challenges. The loss of data and data breach violates two security requirements: integrity and confidentiality. Here, data loss refers to the state of users' data damaged or skipped due to physical means used during processing, transmission, or storage. In a data breach situation, users' data is stolen or copied by an unauthorized user. These two types of attacks can be carried out by both malicious insiders in the cloud environment or malicious applications that resides on the user device.

Another issue to consider is data recovery. This is the process of recovering data from mobile users' data that has been damaged, failed, corrupted, or lost, or from physical storage devices. However, because users' data is stored on the service providers' premises, users require knowledge of the location or storage of their data, and thus data locality is a challenge. Additionally, users' data must be stored separately from other data. If one user's data is mixed, combined, or confounded with the

data of other users, it becomes significantly more vulnerable. When data is outsourced to cloud servers to increase storage capacity, mobile users simultaneously lose physical control of their data. Thus, in a cloud storage scenario, data accuracy becomes a concern for mobile users. Although cloud infrastructures are far more reliable and robust than mobile devices, they still face many threats to data integrity from both internal and external sources.

2.2.2 Partitioning and Offloading Security Issues

Access to the cloud via wireless networks is required during the offloading process. Due to mobile users' lack of control and access over their offloading processes, there is a risk of unauthorised access to offloaded content. Additionally, because offloaded content is executed on cloud or edge servers rather than mobile devices, offloaded content's integrity and confidentiality are possibly violated. The integrity issue arises because, following the execution of offloaded content, mobile devices cannot easily verify the results' correctness if the result is incorrect or altered. Other difficulties, however, include attacks on availability and malicious content threats. Jamming attacks between data/application and mobile device during partitioning and between mobile device and cloud during offloading can jeopardise cloud service availability. Additionally, the presence of malicious content between the partitioning and offloading stages can jeopardise the confidentiality of users' data and violate mobile users' privacy.

2.2.3 Virtualization Security Issues

Cloud service providers in MCC provide cloud services to mobile users via virtualization techniques. At the cloud end, an image of the mobile device's virtual machine (VM) is pre-installed, and the mobile device's tasks are offloaded to the VM for processing. This virtual machine is also referred to as a thin VM or a phone clone. The primary purpose of virtualization is to enable multiple virtual machines (VMs) on a single physical machine or mobile device while keeping the VMs isolated from one another. An additional layer known as a hypervisor or VM Monitor, or Manager (VMM) is software that enables the creation, operation, and control of virtual machines (VMs) and their associated virtual subsystems. However, when applied to MCC, virtualization techniques introduce several security challenges (Sgandurra and Lupu, 2016), including security challenges within VMs, unauthorized access, VM-to-VM attack, communication security within the virtualized environment, security challenges within the Hypervisors, and data confidentiality.

2.2.4 Mobile Cloud Applications Security Issues

As the number of MDs used in the MCC environment grows rapidly, developers are creating a wide range of mobile apps for these devices. Most of these apps offer cloud-based services with rich user experiences. With the help of these mobile apps, MCC users can access rich cloud-based services and experiences on even low-resource MDs. These apps must scale quickly to meet both the user requirements and device capabilities (Mollah et al, 2017).

Attacks on cloud-based mobile applications can compromise the integrity and confidentiality of both data and applications through various strategies, including the integration of malware (Pokharel et al., 2017; Prokhorenko et al., 2016; Peng et al., 2016; Quick and Choo, 2016). Malware, viruses, worms, trojans, rootkits, and botnets (Arabo and Pranggono, 2013) are malicious, contrary, intrusive, and obstructive applications or programmed codes. This malware is designed to run maliciously on mobile devices or to attach to applications without the user's consent. As a result, mobile application functionality can be altered. An attacker will identify a target application, inject malicious code into it, and then republish it. A further discussion of the security solutions for mobile cloud application and data in the context of this study can be found in sections 2.5.1 and 2.8.

2.2.5 Mobile Device Security Issues

Physical threats to mobile devices exist. If mobile devices are misplaced, lost, or stolen, data or applications may be lost, leaked, accessed, or unintentionally disclosed to unauthorised users (Milligan and Hutcheson, 2008). Although many mobile users have password or pattern-based lock features, many do not use them. Additionally, the identity module card within the mobile device can be removed and accessed by unauthorised individuals. Additionally, most MD lack a defence mechanism against threats. The attackers can attack by employing a variety of availability attack techniques, including sending a high volume of malicious traffic and sending large messages to target mobile devices to render them inactive or reduce their capability. Liu et al. (2009) investigate and identify several security mechanisms and critical flaws in security models for intelligent mobile devices.

In addition, the authors demonstrate how to launch a distributed denial-of-service attack by exploiting the vulnerabilities. However, a battery power exhaustion attack is another type of availability attack in which the mobile device's battery power is rapidly depleted following the attack. This attack is unique to mobile devices because it exploits wireless network vulnerabilities, and mobile users are unaware of this type of attack. Racic et al. (2006) discusses this type of attack. They demonstrate here that this attack causes a mobile device's battery to drain up to 22 times faster than it usually does, rendering the device completely useless within a short period. Due to the increasing popularity of mobile devices and applications, malware authors and attackers focus their efforts here. As a result, malware poses a significant security risk to mobile users' privacy, applications, and data.

Additionally, the functionalities of contemporary mobile platforms are pretty similar to those of personal computers. However, they include additional features, and these platforms for mobile devices support a wide variety of applications.

As a result, to maintain the confidentiality and integrity of these applications, it is necessary to secure the mobile platforms. Additionally, these mobile platforms are not malware-free. Typically, attackers gain root permissions on mobile devices and gain control of the device, after which they can directly affect the computational integrity of mobile platforms and applications. In mobile devices, three types of storage are available: on-device storage, plugged-in storage, and identity module storage. Generally, these storages are used to store users' data, applications, and other types of data. However, if a mobile user utilises cloud services, the user's data and applications are replicated in cloud storage. Thus, if a mobile device is stolen or lost, it becomes critical because attackers can access both the mobile device and the cloud.

2.2.6 PRIVACY ISSUES

Privacy is a significant issue because confidential data or applications of mobile users are processed and transferred from mobile devices to heterogeneous distributed cloud servers while utilising various cloud services. These servers are in various locations and are solely owned and maintained by the service providers. Because users cannot physically justify their data storage, data privacy and protection issues are left to service providers, and users are not held accountable for privacy breaches. Cloud storage and processing in multiple locations create privacy concerns. Service providers' cloud servers are in various regions and countries. For example, Google's cloud servers are spread globally, with seven locations in the Americas, two in Asia, and three in Europe.

Additionally, it is critical for users to obtain information about the cloud hosting location, as laws vary by country. Numerous mobile applications may be unsafe due to their hideous functions, the unintentional collection of users' personal information such as hobbies and locations, and the potential for illegal distribution. Unwanted advertising emails, also known as junk emails, can infringe on users' privacy.

Context awareness, enabled by sensors on mobile devices, is one of the primary characteristics that differentiate mobile applications from personal computers. The context informs service providers by providing context for users, allowing service providers to tailor their offerings to their specific needs. These location-aware applications and services raise concerns about mobile device privacy. These can be user-initiated or service provider-initiated and require the user's location to provide location-based services. Additionally, many applications require and collect users' location data, which they can use to target clients directly based on their locations. As a result, location-based services present privacy concerns due to collecting, storing, and processing user data.

2.3 MOBILE CLOUD COMPUTING SECURITY VULNERABILITIES

The NIST Security Glossary defines vulnerabilities as "weaknesses in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source." (Kissel,2011). Grobauer et al. (2010) demonstrated how flaws in enabling technologies lead to vulnerabilities in CC architectural components. Numerous researchers have researched the vulnerabilities in CC architecture components. For example, Fernandes et al. (2014) detailed the threats, vulnerabilities, and attacks against cloud infrastructure in their work. Hashizume et al. (2013) discussed various cloud system vulnerabilities and proposed associated countermeasures. The following subsections discuss some security vulnerabilities in the CC environment that are also present in the MCC environment.

2.3.1 VULNERABILITIES IN APPLICATION AND INTERFACE LAYER

This layer serves as the gateway to a cloud provider's services, typically accessed via the Internet. An MCC user typically accesses the cloud services via a web browser or mobile application on their MD. Therefore, this layer is as vulnerable as web technologies' flaws and weaknesses and security issues in the MD environment, such as malicious apps installed on the user device and the Internet (Rittinghouse & Ransome, 2017; Jensen et al., 2009).

Apart from confidentiality and privacy, one of the fundamental security requirements that this layer must meet is cloud user authentication and authorization. The Open Web Application Security Project's (OWASP) list of the ten most serious web application security threats also applies to the cloud computing environment (OWASP,2021). These include Broken Access Control, Cryptographic Failures, Injection, Insecure Design, Security Misconfiguration, Vulnerable and Outdated Components, Identification and Authentication Failures, Software and Data Integrity Failures, Security Logging and Monitoring Failures, and Server-Side Request Forgery. Due to the unique characteristics of CC, the conventional approach to addressing these vulnerabilities will be insufficient. A novel approach to addressing vulnerabilities associated with web technologies will need to be adopted. As PaaS and IaaS services require management interfaces and applications for users accessed via web services and interfaces, these management interfaces also have the same level of vulnerability as SaaS applications (Grobauer et al., 2010).

Web services and interface vulnerabilities can result in data leakage and unauthorised access to resources. Grobauer et al. (2010) identified cross-site scripting, command injection, and SQL injection as mechanisms for manipulating service requests to exploit the vulnerability in web services interfaces. Additionally, they stated that a malicious agent is likely to steal the credentials of a web service requester due to a faulty implementation of the session handler, which results in session hijacking and session riding.

Access to read and modify web browser components during transactions between client applications of a cloud user and the cloud provider's web application server results in client-side data manipulation vulnerabilities. When attacks are launched against this class of vulnerabilities, information confidentiality and integrity are jeopardised. The injection is ranked as the top three security risk in OWASP's Top 10 Application Security Risks - 2021, indicating that data manipulation vulnerabilities at the web client are the primary weakness in providing a secure CC and MCC environment.

The most common method of exploiting the injection vulnerability is SQL injection, which is accomplished by injecting a valid parsing string with malicious intent into the web client's original legitimate SQL query request to the web application server via the provided application interface. The HTTP hidden fields, which are typically used to store a web user's login information via web forms, are another source of data manipulation vulnerabilities, as attackers can steal user credentials via a watering hole attack (falling for a duplicate or fake, fraudulent website). The growing popularity of social networking sites exposes the user's browser to self-installing malware. It is a significant source of data manipulation vulnerabilities for web browsers due to the possibility of exploiting user credentials entered by users.

The cloud's on-demand nature and multi-tenancy make identity management, authentication, and authorization processes vulnerable. Identity management (IDM) is a broad term that refers to the process of identifying entities, including cloud objects, and enforcing policies that restrict access to resources. Authentication vulnerabilities occur due to insecure user behaviours such as using weak passwords, reusing credentials, relying on one-factor authentication, and having a poor credential life cycle management process. Inadequate authorization checks and a lack of control over user privileges result in authorization vulnerabilities. Fernandes et al. (2014) noted in their survey that even graphical, biometric, and three-dimensional passwords have limitations. Numerous studies have identified flaws in authentication and authorization protocols. OpenID Connect, like OAuth2.0, is widely used by web applications as a single sign-on (SISO) mechanism for end-user authentication.

2.3.2 VULNERABILITIES IN PLATFORM LAYER

The platform layer provides development and deployment tools, middleware, and operating systems to enable PaaS offerings to cloud users to develop and deploy their custom applications. The vulnerabilities determine the layer's security level in custom software and the operating system. Real-time software applications must pass functional and security checks before they are used to enhance security of this layer (Fernandes et al., 2014).

The quality of the software is highly dependent on the software development framework used by programmers throughout the software development lifecycle. Frequently, fundamental design and

development practices are undermined in the name of project time constraints. Vulnerabilities occur due to insufficient and incomplete verification and validation of the software deployed at the platform layer. Often, security concerns are overlooked or ignored during the software development life cycle (Rittinghouse & Ransome,2009).

Security issues in a software application are typically caused by vulnerable programming codes, which explains why exploitation has increased. Programmers who violate the best coding practices and guidelines introduce vulnerabilities into their code (Rittinghouse & Ransome,2009). Rodero-Merino et al. (2012) raised security concerns about the Java and.NET platform development environments in instances of unsafe thread termination and violation of memory zone isolation. When untrusted third-party software, primarily open-source code, is deployed on a cloud platform, the platform becomes vulnerable to various security attacks (Fernandes et al.,2014). The cloud application's back-end server code is susceptible to malicious masked code injection (such as SQL injection) via a request from the front-end, such as a web browser.

Through system calls, operating systems installed in a virtual machine facilitate communication between applications and hardware. As a result, it has access to all data within a virtual machine. As a result, any malicious services that run in the background could result in data leakage. A malicious system administrator can bring the entire operating system software to a halt. Inappropriate resource allocation and monitoring have a detrimental effect on the performance and availability of the system. Inadequate memory isolation can result in data leakage. Incomplete and insufficient monitoring of the operating system results in unnoticed malicious actions (Babu & Bhanu, 2015).

2.3.3 VULNERABILITIES IN INFRASTRUCTURE LAYER

The communication within a cloud network can be classified into external and internal (Ali et al.,2015). External communication occurs outside of the cloud, i.e. between cloud components and user, via the Internet. Internal communication occurs between cloud components and virtual machines via virtual network communication channels. Thus, the cloud network infrastructure is inherently vulnerable to protocols and technologies associated with the Internet and virtual networks (Fernandes et al., 2014).

These are security flaws inherent in the network communication protocol and technology used to access cloud services. Protocols based on the TCP/IP stack, such as DHCP, IP, and DNS, are vulnerable to IP spoofing, DNS cache poisoning, and DNS spoofing and may result in cross-tenant attacks (Almorsy et al.,2016). When a malicious website is visited, an attacker may alter the DNS settings on the user's broadband network router. Inadequate implementation of session management techniques to deal with HTTP statelessness may result in session riding or hijacking (Grobauer et al., 2010). When HTTPS and HTTP are used in conjunction with cloud services,

communication between web clients and cloud services becomes insecure (Prandini et al., 2010). According to Jensen et al. (2009), flooding attacks pose a real threat to the cloud, resulting in direct DoS, indirect DoS, and accountability issues.

Online storage services are fascinating due to their large storage capacity, high availability, and stable performance. At the same time, it introduces security risks due to a lack of transparency and direct control over data stored in a cloud environment (Aguiar et al.,2014). Additionally, cloud-specific characteristics such as virtualization and multi-tenancy create inherent security challenges for cloud-based data storage (Ali et al., 2015). Chen and Zhao (2012) discussed various vulnerabilities related to the data lifecycle in a cloud environment. Singh et al. (2016) discussed storage security concerns regarding cryptography, data persistence, data sanitization, data leakage, malware snooping, and availability.

Data storage is vulnerable to security attacks due to poor key management, faulty, insecure, and obsolete encryption algorithms (Grobauer et al.,2010). Modi et al. (2013) survey identifies a poor encryption technique as the primary risk. Data stored in a cloud environment is perpetually vulnerable to tampering by outsiders and insiders (Sood,2012). Due to the shared environment, compromised keys, and application vulnerabilities, data stored in cloud storage (data-in-rest) is vulnerable to unauthorised access (Modi et al.,2013).

The CC environment is generally distributed across multiple geographical locations to maximize cost-effectiveness, scalability, redundancy, and disaster recovery. Local legal and regulatory policies impact the security and privacy of user data. Cloud users are concerned about the physical location vulnerability of the data centre that houses the storage and its backup. Unauthorized access and tampering are possible with a backup storage. While the cloud's resource pooling and elasticity characteristics enable dynamic resource allocation and sharing, they also present a unique security challenge regarding data recovery vulnerabilities exploited by the newly allocated user. Cloud storage is not immune to backup data recovery (Modi et al.,2013).

2.4 Mobile Cloud Computing Security Requirements and Threats

Solving security issues in MCC requires identifying the security requirements and associated threats resulting from vulnerabilities in the MCC environment that can lead to possible attacks on its resources. The NIST in 2011 stated that confidentiality, integrity, and availability are the basic security requirements of CC, just like in any other information security system (Liu et al., 2011). CSA, one of the top organizations providing security guidance to the CC community, adds another four security requirements (authentication, authorization, accountability, and privacy) to the three basic ones as shown in Table 2.1 (Mogull et al.,2017). These seven basic requirements are also applicable to the MCC environment.

The NIST security glossary defines a threat as ‘‘Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or the Nation through an information system via unauthorized access, destruction, disclosure, modification of information, and denial of service’’(Kissel, 2011). CSA has adopted these definitions in their threat analysis concerning the cloud ecosystem and the proposed security solution guideline reported between 2010 and 2019. Taking account of Kumar & Goyal (2019), their study uses “The Treacherous Twelve top threat in cloud security” as a baseline to identify vulnerabilities in the cloud architectural framework. In this study, the egregious eleven top threats in cloud security reported by the CSA in 2019 is adopted to analyze existing data security frameworks in the review of related works. The CC security threat analysis reported by the CSA between 2010 to 2019 is presented in Table 2.2. The description of the egregious eleven cloud security threats used to investigate existing data security frameworks in MCC is shown in Table 2.3.

Table 2.1. The Seven Basic MCC Security Requirements

ID	Security Requirement	Description
R1	Confidentiality	Sensitive data of users should be kept secret and not accessible by an unauthorized user.
R2	Integrity	Protection of MCC user data from modification or deletion without authorization.
R3	Availability	MCC user data and services should be available for access at any time whenever the user demands.
R4	Authentication	Every user in the MCC environment needs to have their identity verified before accessing cloud services.
R5	Authorization	Access control rights to each MCC resource must be properly defined for each user.
R6	Accountability	CSP must establish all-action administered in its cloud environment to a single entity either to the cloud user, the process, or the mobile device must be done in a legitimate fashion.
R7	Privacy	CSP must ensure that MCC user data are not used for any purpose without the authorization of the data owner.

Table 2.2. The evolution of the CC security threats

Rank	2010	2013	2016	2019
1	Abuse of Cloud Services	Data breaches	Data breaches	Data breaches
2	Insecure APIs	Data loss	Weak identity, credential, and Access management	Misconfiguration and Inadequate Change Control
3	Malicious insiders	Account hijacking	Insecure APIs	Lack of Cloud Security Architecture and Strategy
4	Shared Technology vulnerabilities	Insecure APIs	System and Application Vulnerabilities	Insufficient Identity, Credential, Access and Key Management
5	Data loss/Leakage	Denial of Service	Account hijacking	Account hijacking
6	Account, Service & Traffic hijacking	Malicious Insiders	Malicious Insiders	Malicious Insiders
7	Unknown Risk Profile	Abuse of Cloud Services	Advanced Persistent Threats (APTs)	Insecure APIs
8		Insufficient Due Diligence	Data loss	Weak Control Plane
9		Shared Technology Issues	Insufficient Due Diligence	Metastructure and Applistructure Failures
10			Abuse of Cloud Services	Limited Cloud Usage Visibility
11			Denial of Service	Abuse of Cloud Services
12			Shared Technology Issues	

Table 2.3. The Egregious eleven CC Threats

ID	Threat Name	Threat Description
T1	Data breaches	This is an incident that results in a release, access, stolen or use of protected sensitive information by an unauthorized user.
T2	Misconfiguration and Inadequate Change Control	The absence of effective change control in the CC environment leads to a misconfiguration of cloud resources which makes them vulnerable to malicious activities and data breach.
T3	Lack of Cloud Security Architecture and Strategy	Inadequate understanding of shared security responsibilities during the migration of IT services to the public cloud without an effective security architecture to withstand cyber-attacks.
T4	Insufficient Identity, Credential, Access and Key Management	This consist of tools and policy that allow the organization to manage, monitor and secure resources within their infrastructure. Incident usually occurs if either party (CSP and cloud user) comprise their security as a result of weak credential protection, lack of scalable identity and credential access management, failure to use multi-factor authentication and strong password policy.
T5	Account hijacking	Loss of device or unauthorized access to device credentials can be used to hijack user account, which leads to illegal access to cloud resources of the mobile user.
T6	Malicious Insiders	An authorized user of the cloud services can intentionally lunch attacks against the cloud resources through legitimate access to these resources.
T7	Insecure APIs	APIs allows third-party agents to interact with the cloud services; vulnerabilities in these APIs expose these resources to the external world and malicious attacker for possible attacks.
T8	Weak Control Plane	This Incident results in data corruption, unavailability, or leakage due to the weak nature of the control plane for data duplication, migration strategy and storage, especially when migrating to a multi-cloud infrastructure. This weak nature of the control plane makes the person in charge not have full control of data infrastructure logic, security, and verification.
T9	Metastructure and Applistructure Failures	Poor APIs authentication designed by the CSP enables malicious attackers an opportunity to comprise the security of the CC resources. This is because API calls disclose sensitive information, which is incorporated in the Metastructure, which is a line of demarcation between the CSP and the cloud user.
T10	Limited Cloud Usage Visibility	This is a type of threat that occurs when an organization cannot analyze the existing cloud-based application used within its organization to determine if the application is safe or not. Such an employee using an application without the permission of the IT security personnel can compromise organization CC services which can result in credential theft, Structured Query Language (SQL) Injection, domain name system attacks e.t.c
T11	Abuse of Cloud Services	This kind of threat to the CC environment enables an attacker to exploit vulnerabilities in the cloud deployment strategy such as poor security policy and lunch attacks such as email spamming, phishing campaigns, and denial of services attacks against the cloud resources.

2.4.1 MCC THREAT TREND ANALYSIS

There has been a great change in the CC threat trends in recent times, as shown in Table 2.2 and Table 2.3. The recent report, as published by CSA in 2019, took a new direction, as new threats emerge (i.e., Misconfiguration and Inadequate Change Control, Lack of Cloud Security Architecture and Strategy, Weak Control Plane, Metastructure and Applistructure Failures and Limited Cloud Usage Visibility). There is a shift from a focus on traditional information security threat focus to a focus on CC Infrastructure configuration and authentication issues, as shown in Figure 2.1.

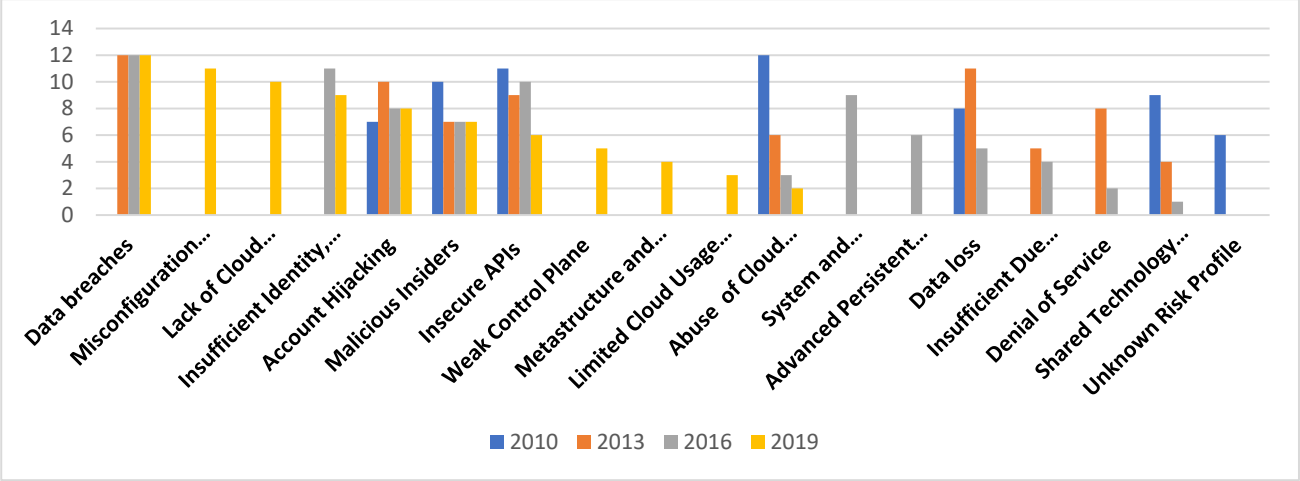

Figure 2.1 CC Threat Ranking Change Spectrum

Figure 2.1 shows the ranking of different threats from 2010 to 2019. In 2013, three new threats were added while the unknown profile risk was removed. Similarly, in 2016 three new threats: Weak Identity, Credential and Access Management, System and Application Vulnerabilities and Advanced Persistent Threats (APTs) were reported. The recent report released by CSA in 2019 show a new ranking of Abuse of Cloud Services: it was placed at the top of the list in 2010 but was ranked 11th in 2019. These changes also indicate that, despite the wide acceptance of CC in recent years, the CC infrastructure credibility and the trust in its use have been questioned. For example, data breaches have been ranked as the topmost threat in three consecutive times (i.e., 2013, 2016 and 2019).

2.4.2 THE EGREGIOUS CC THREAT ANALYSIS MAPPING USING THE STRIDE MODEL

Scandariato et al. (2015) describes the threat modelling tool (STRIDE) as reported by Microsoft in 2009. The STRIDE model reported by the study conducted by Scandariato et al. in 2015 evaluated an Information System's security weakness using a descriptive approach. Similarly, this study adopted the STRIDE threat modelling method to investigate the existing data security frameworks in the MCC environment in the subsequent section of the literature review and presented the threat analysis report using the STRIDE model reported by the CSA in 2019, as shown in Table 2.4. The

STRIDE model allows security experts to identify and develop solutions to CC and MCC security issues. The STRIDE model includes six threat categories that are briefly described as follows:

A. Spoofing Identity (S): This threat category describes the use of stolen credentials to gain unauthorized access to an information system or cloud resources. This threat category affects the Authentication (R4) basic security requirements in CC and MCC environments.

B. Tampering with Data(T): This threat category describes the malicious modification of data during transmission and at rest by an unauthorized user. This specific threat category affects the integrity of user data(R2) in both CC and MCC environments.

C. Repudiation (R): This threat category describes a denial of malicious activities carried out by the user without any means of verifying the actual user that carried out such illegal activities in both the CC and MCC environment. This threat category affects the accountabilities(R6) basic security requirements of this environment.

D. Information Disclosure(I): This threat category describes the exposure of user information or access to user store data by an unauthorized user. This threat category affects both confidentiality(R1) and privacy(R7) of user data in CC and MCC environments.

E. Denial of Service (D): This threat category affects the availability (R3) of cloud resources to the legitimate user by denying the user access to critical cloud resources at the time of their request.

F. Elevation of Privilege(E): This threat category grants excessive privilege gain by the user without authorization. This threat category affects the Authorization (R5) basic security requirements of both the CC and MCC environments.

Table 2.4: STRIDE Threat Analysis using the Egregious CC Threats

Threat ID	S	T	R	I	D	E
T1	×	×	×	✓	×	×
T2	×	✓	✓	✓	✓	×
T3	✓	✓	✓	✓	✓	✓
T4	✓	✓	✓	✓	✓	✓
T5	✓	✓	✓	✓	✓	✓
T6	✓	✓	×	✓	×	✓
T7	×	✓	✓	✓	×	✓
T8	×	✓	×	✓	×	✓
T9	✓	✓	✓	✓	✓	✓
T10	✓	✓	✓	✓	✓	✓
T11	✓	✓	✓	✓	✓	✓
✓ Indicates that the threat affects the threat category of the STRIDE model						
× Indicates that the threat does not affects the threat category of the STRIDE model						

2.5 Analysis of Current MCC Data Security Frameworks

To address the main research question and its sub research questions, following the specific research objectives defined in chapter one, section 1.3. This section of the thesis addresses the first research objectives by analysing data security frameworks proposed and used in recent MCC security research. The search for relevant data security frameworks was conducted across four electronic databases (IEEE, Science Direct, ACM, and Springer) from 2010 to 2021. To investigate the current MCC data security solution scope in-depth, only relevant security frameworks that address core data security issues in the MCC domain were selected. This study selected and analysed thirty-five (35) peer-reviewed articles published in conferences and journals between 2010 and 2021 to address the first research objective.

The selected MCC data security framework presented in Table 2.5 was analyse using the following predefined set of dimensions:

Domain (D1): This dimension describes the proposed security framework's architectural layers (MCUL, MNCL, and MCSPL).

Threats (D2): This dimension describes the threats that the proposed framework addresses using the eleven threats (T1 to T11) presented in Table 2.3.

Security Requirements (D3): This dimension describes the security requirements that the proposed framework addresses, as presented in Table 2.1.

Security Approach (D4): This dimension describes the proposed framework's techniques to protect user data (such as encryption, biometrics, access control, activity monitoring, and intrusion detection).

Cloud Level Trust (D5): This dimension identifies the MCC components that are trusted by the framework. This study assumes that using a trusted third party within a proposed framework shows that the cloud infrastructure is not trustworthy. Similarly, if there is no third-party component in the proposed framework, the cloud infrastructure is assumed to be trusted.

Comprehensiveness (D6): This dimension is used to measure the number of threats a framework addresses. A framework is considered more comprehensive in this study if it addresses five or more threats; otherwise, it is less comprehensive.

Table 2.5: Analysis Summary of the Selected Data Security Frameworks

ID	Source	D1	D2	D3	D4	D5	D6
F1	Itani et al, 2010	MCUL/ MCSPL	T1	R2	DE,DV	NT	
F2	Huang et al. , 2010	MCUL/ MCSPL/MNCL	T1-T4,T11,	R1,R4,R5	TM	T	
F3	Jia et al., 2011	MCUL/ MCSPL	T1-T2,T4	R1,R4,R5,R6,R7	DE,DV	T	
F4	Huang et al.,2011	MCUL/ MCSPL/MNCL	T1,T2,T4	R1,R4,R7	TM	NT	
F5	Chen & Wang, 2011	MCUL/ MCSPL	T1,T4,T8	R1,R3,R4,R7	LBS	NT	
F6	Lin,2011	MCUL/MCSPL	T1,T3,T4,T8,T10	R1,R3,R4,R5,R7	DM	NT	*
F7	Lu et al., 2012	MCUL/ MCSPL	T1,T4	R1	DE	NT	
F8	Omri et al., 2013	MCUL/ MCSPL	T1,T4-T5	R1,R4,R5,R7	BM,DE	NT	
F9	Khan et al., 2013	MCUL/ MCSPL	T1,T4	R1,R4,R5,R7	TM,DE	T	
F10	Zhang et al., 2013	MCUL/ MCSPL	T1,T5,T8	R1,R3,R7	TM	NT	
F11	Khan et al., 2014	MCUL/ MCSPL	T1,T4	R1,R4,R5	DE	T	
F12	Dey et al., 2015	MCUL/ MNCL	T1,T3-T4,T7-T9,T11	R1,R3,R4,R7	IDS,DE	T	*
F13	Shi et al., 2015	MCUL	T1,T3-T8,10,T11	R1,R3,R4,R7	IDS,DE	T	*
F14	Goyal & Krishna, 2015	MCUL/ MCSPL	T1,T4,T5,	R1,R4,R5,R7	DE	T	
F15	Shiny et al., 2015	MCUL/ MCSPL	T1,T4,T5	R1,R4	DE	T	
F16	Benabied et al.,2015	MCUL/ MCSPL	T1,T3-T6,T8,T11	R1,R3,R4,R7	TM	NT	*
F17	Thumar & Vekariya, 2016	MCUL/ MCSPL	T1	R1,R7	DE	T	
F18	Zhang & Wen, 2016	MCUL/ MCSPL	T1,T5,T11	R1,R4,R7	DE	NT	
F19	Cushman et al., 2017	MCUL	T1,T7,T9	R1,R7	DV	T	
F20	Lin et al. ,2017	MCUL/ MCSPL	T1,T7,T9	R1,R2,R7	DE	NT	
F21	Arvind & Manimegalai, 2017	MCUL/ MCSPL	T1	R1,R2,R7	DE	NT	
F22	Khatri & Vadi, 2017	MCUL/ MCSPL	T1,T4,T5	R1,R4,R7	BM	NT	
F23	Sajjad et al., 2017	MCUL/ MCSPL	T1	R1,R7	DE,ST	NT	
F24	Li et al., 2017	MCUL/ MCSPL	T1,T4,T5	R1,R4,R7	DE	NT	
F25	Chean et al , 2018	MCUL/ MCSPL	T1,T4,T5	R1,R4,R7	DE,TM	NT	
F26	Nguyen et al., 2018	MCUL/ MCSPL	T1,T3,T5,T7-T9	R1,R3,R7	IDS	T	*
F27	Sukumaran & Mohammed , 2021	MCUL/ MCSPL	T1,T4,T5	R1,R2,R4,R7	DE	T	

ID	Source	D1	D2	D3	D4	D5	D6
F28	Dey et al. , 2019	MCUL/ MCSPL	T1,T3,T5,T7-T9,T11	R1,R4,R7	IDS	NT	*
F29	Agrawal & Tapaswi , 2019	MCUL/ MCSPL	T1,T4,T5	R1,R4,R7	DE	T	
F30	Nguyen et al., 2019	MCUL/ MCSPL	T1,T4,T5,T8,T11	R1,R2,R4,R5,R7	BC	T	*
F31	Khedr et al., 2020	MCSPL	T1,T4,T5	R1,R4	DV	T	
F32	Irshad et al., 2020	MCSPL	T1,T4,T5,T6	R1,R3,R4	DE,DV	T	
F33	Derhab et al., 2020	MCUL/ MCSPL	T1,T2,T4,T5,T11	R1,R4,R7	DV	T	*
F34	Li et al.,2020	MCSPL	T1,T4,T5	R1,R4,R7	DE	NT	
F35	Shabbir et al.,2021	MCSPL	T1,T4,T5,T6	R1,R4,R7	DE	T	

***Note**:* DE- data encryption; DV – data verification, Block Chain, BM – Biometric; IDS – intrusion detection system, TM- trust management, DM- data mining, LBS-location based services, ST-steganography (*) - “more comprehensive” (T) –“Trusted” (NT) –“Not Trusted”

2.5.1 Review of the MCC Data Security Frameworks

The description of each of the MCC data security frameworks used for the analysis in this review are discuss as follow:

The framework proposed in F1 uses an incremental message authentication code to guarantee data storage integrity in the MCC environment. The framework protects MCC user data while subduing the device's energy consumption level and efficiently supporting dynamic data operations using trusted computing. The model proposed in F1 comprises three entities: 1) mobile client, 2) cloud service provider, and 3) trusted third party. The model allows the offloading of the verification job to a coprocessor on the cloud end to reduce the processing overhead from the MD layer. The integrity of mobile data is achieved by generating an incremental authentication code using a shared key before uploading data to the cloud. The code is stored locally on the MD, and the data is offloaded to the cloud. For data processing operations to be carried out on the offloaded data or files, there is a need to perform an integrity check for the file using the message authentication code.

The proposed framework (MobiCloud) in F2 addresses trust management, risk, and secure routing in ad hoc networks. This framework converts traditional mobile ad hoc networks into a new service-oriented communication structure in which each MD is handled as a service node (SN). The device produces different copies of extended semi-shadow images (ESSIs) in the cloud, which provide a solution to an MD's communication and computation requirements. Data received from the MD requires users to trust the service provider of the cloud infrastructure for the security of the data.

The authors in F3 proposed a secure data service mechanism (SDSM) that gives MCC users the tool to enjoy secure data outsourcing in a trusted mode. This framework allows users to benefit from

minimized overheads associated with security management in the MCC environment. The framework (F3) comprises two models, namely the network and security models. The network model protects data leakage from a third party because the data owner uploads encrypted files on the cloud server for use. The data owner must provide authorization for other cloud users, such as data sharers, to access the files of the data owner. The authorization permission granted by the data owner allows the data sharer to decrypt the specific files in the MCC environment. The security model employs a proxy re-encryption scheme and a bi-linear mapping of an identity-based encryption model to protect data from malicious sharers in the data sharing process in the cloud environment. The framework in F3 also addresses the collusion attack of malicious data sharers that might affect the secrecy of the data owner's secret key. This framework's security model was used to realize strong data access control, low overhead processing, and flexibility to operate in a highly scalable environment.

The authors in F4 address the privacy and security issues of MobiCloud by proposing a framework for data processing for mobile cloud users using trust management, multi-tenant secure data management, and the ESSI processing model. The framework isolates private data using a trust management approach. The framework consists of three components: 1) the cloud mobile and sensing domain, 2) the cloud as the trusted domain, and 3) the cloud public service and storage domain. The Framework (F4) also supports the virtualization of each mobile device in a specific application domain. Their virtualization techniques in this framework address a mobile device's communication and computation issues and help improve security and privacy protections. The FocusDrive project was designed as a prototype to demonstrate the security and privacy protection techniques as proposed in their framework(F4).

The proposed security framework in F5 enhances the privacy and authentication of mobile device users using location-based services in the cloud environment. The framework (F5) uses distributed storage and international mobile subscriber identity to improve the security of user data. The result evaluation of this framework shows that the network coding scheme has improved performance. This framework (F5) enhances the security issues caused by multiple tenants and multiple replicas of mobile user data in the cloud environment.

The MCC security frameworks in F6 investigate state-of-the-art technology in mobile security in CC environment. The authors' results in F6 show that most of the existing mobile security frameworks allow files to be sent to the cloud for processing due to the resource constraint of the MDs. This approach faces critical issues such as compromised user data security and privacy, high network load, high time consumption in processing, low connectivity, and bandwidth in securing MD files. To provide a solution, the author proposed a new "Private Cloud and File Characteristics" (PCFC) framework that has three (3) components: mobile client, private cloud, and PCFC protocol. The

mobile client is a lightweight application that runs on the MD that uses a specific extraction algorithm to extract specific data from a suspicious file for processing in the cloud. The proposed framework in F6 helps to solve some of the issues some of the existing frameworks faced. The private cloud uses a scoring process to effectively identify files with malicious data and apply appropriate measures to such a file using the protocol.

The framework in F7 proposed a dynamical data protection framework for off-trade between security and resource consumption using a service access gateway. The model has a database of security policies, and mobile users get assignments of data protection services for each requesting terminal. In this framework (F7), different data protections are applied to data depending on the services policy request by the mobile terminal that initiates the request.

The authors of F8 propose a framework that uses handwritten password recognition services to authenticate users in the mobile cloud environment. The model implements an application that creates an interface between the mobile device and the cloud, which captures handwritten passwords from the mobile device as biometric data, encrypts the handwritten password data, and transfers it to the cloud using the mobile device touch screen. Biometric data encryption enhances the security and privacy of data access in the mobile cloud environment. The cloud end contains a database of encrypted handwritten templates of cloud users to perform authentication. The authors of F8 use K-nearest neighbour and an artificial neural network classifier for the handwritten recognition of user passwords. The Framework authenticates the user based on the password and biometric features using handwritten password techniques. The classifier algorithm uses a parallel classifier combination method to obtain satisfying recognition and error rate precision.

F9 proposed a lightweight security framework to safeguard the mobile user's identity with dynamic credentials to enable the identification of users in the mobile cloud environment. The framework offloads the regularly occurring dynamic credential generation services into a trusted environment to reduce the processing overhead, delay in communication, and loss of energy on the mobile device. The system model used in this framework has the following components: 1) cloud service provider, 2) mobile users, and 3) trusted entity. Computational and storage services are offered to mobile users by the cloud services provider. Mobile users employ those services to improve their MDs' processing and storage inclinations. The trusted entity is responsible for generating cloud secrets, mobile secrets, and credentials for the mobile user under the control of the client organization. The trusted entity securely hands over the prevailing dynamic secrets to the cloud service provider and mobile device to identify mobile users in a cloud environment. The trusted entity also shares each mobile user's private and public keys and the CSP. The results obtained from the experiment performed for this framework show a significant improvement in energy consumption on mobile devices compared with other frameworks. The framework (F9) works in an untrusted environment,

reducing the possibility of a man-in-the-middle attack. The framework verifies the reliability of the credentials of mobile users with the help of the received signature, and the user credentials are encrypted with the mobile user's public key that ensures confidentiality.

The authors of F10 proposed a framework that uses a decentralized data aggregation approach for securing cloud user data. The framework allows distributed computing among mobile devices, personal clouds, and public clouds. In this framework, user data is stored in a private cloud in which the user has full control. Data widget applications were introduced for the user to manage sensing data. The framework (F10) addresses the privacy issues of mobile sensing systems by using a trusted third-party certificate to monitor the behaviour of applications developed by cloud users. These trusted party security techniques minimized the issue of publishing malware in the cloud computing environment.

The proposed framework in F11 provides an incremental proxy re-encryption scheme to improve file modification while maintaining the security of the offloaded file in the cloud infrastructure. The framework enhances processing overhead and communication delay. The scheme uses bi-linear mapping for randomly generated parameters for encryption, decryption, and re-encryption of data. Key generation in this scheme uses a trusted entity called a proxy to create key pairs for various authorized group members of the data partition for cloud storage. The incremental version of the proxy re-encryption framework presents significant enhancement in results while implementing file modification operations using the insufficient processing ability of mobile devices in the CC environment.

F12 presented a context-aware security framework for a mobile cloud that requires deployment at the cloud end to mitigate against attacks on user data as an extra security layer to the existing mobile cloud security infrastructure. The framework analyses incoming traffic using a cognitive learning model to identify patterns from previous attacks. The probabilistic model evaluates the packets received from the network traffic to detect possible attacks on the infrastructure. A self-healing network is used to deter denial of service attacks on the cloud side. Authentication ensures mutual authentication between two communication parties using the message digest and location-based services.

F13 presented a cloudlet mesh-based security framework that improves sensitive data protection on mobile devices against malicious activities using intrusion detection techniques in a trusted environment. With the aid of the cloud, the cloudlets update their malware database. Furthermore, a trusted connection is placed among MD, cloudlets, and the remote cloud, and an inter-cloudlets protocol is added to facilitate distributed malware detection. The framework (F13) provides authentication between the cloudlet mesh and the mobile device using a multi-party authentication protocol (MAP). The MAP process requests MD to access the cloudlet mesh. The cloudlet with the

highest signal strength is used for the connection. The MAP process authentication between the mobile device and the cloudlet mesh in a multi-way by scanning only the smaller incoming message using anti-virus software. In comparison, much larger incoming messages are offloaded to the distance cloud for scanning to remove viruses or spam from the request. The multi-party authentication used in this framework is a single sign-on in the cloudlet mesh. The Framework (F13) uses a Trusted Cloud Transfer Protocol (TCTP) to encrypt messages offloaded to the distance cloud.

The authors in F14 presented a security framework that uses encryption technology to secure data transmission. The proposed framework by the authors in F14 uses the identification numbers of the MD to check an intruder whose device is not registered in a central database to enhance data access security using the location-based services model.

The authors in F15 presented a framework that employs digital signature concepts for authentication of received data and encryption for data storage. The framework comprises three phases: 1) key generation, 2) signature generation, and 3) signature verification. The public and private keys used for signing the packets and verifying the packets during encryption and decryption are generated randomly during the key generation phase. Hashing and key generation phase means generating both the public key and private key. The private key is used to sign the packets in the encryption process, and the public key is used to verify the packets in the decryption process. A hashing algorithm is used to produce a hash code. This hash code is obtained from the encryption of the previous operation and the private sender key during the signature generation phase. The signature process is verified after the document has been signed successfully.

The security framework proposed by the authors in F16 presents a two-level security model for the continuous examination of trust degrees. A trust model is used to calculate each user's trust degree and monitor his behaviour and activities. This framework (F16) works with a mobile and a trusted agent. The security of the cloud user and the service provider is considered in this framework to avoid a man-in-the-middle attack against the cloud user data while in transit.

F17 presented a framework that mitigates the security issues cloud users face while maintaining the data in the cloud server and relocating data from the cloud. The framework provides a solution that enables mobile users to store data securely so that the privacy of their data is preserved. The information on the MDs is protected from unauthorized access using a modified RSA encryption technique in which each message is mapped to an integer for preserving data before offloading the data to the CSP storage infrastructure from the MD. This framework provides a software interface for cloud users to perform encryption and decryption. In the first place, the user data is encrypted with the receiver’s public key. After that, the decryption process can be done with the corresponding key pair only.

The authors in F18 presented a framework based on the bilinear pairing cryptosystem and random number theory. It performs user anonymity, mutual authentication, user intractability, and key exchange. In this framework, the cloud service provider is unable to obtain the user's identity. The user's identity is hidden by using a random number to secure sensitive information. The mobile cloud environment is assumed to be supported by a trusted, smart card generator (SCG) service. The SCG is responsible for generating public and private parameters for both cloud service providers and cloud users, respectively. This framework (F18) has four phases: the system setup phase, registration phase, authentication phase, and the passive user phase.

F19 presented a framework that protects data when offloading it from a mobile device to the cloud by verifying if the data to be offloaded has security concerns or not. The security concern is using a mobile API to classify the data into a security class. Once the data classification process is completed, the Mobile API stores and processes data according to rules defined for the data category at hand. The framework uses simulation-based software. The machines used in the framework implementation run on Ubuntu Server 14.04 and have OpenStack cloud software installed to store and provision virtual machines. In the proposed framework reported in F19, the lead node is used as the controller, which manages the communication among the g database storage, software, and permissions among the nodes. The server system uses Ubuntu Metal as a Service (MAAS) to configure software and updates. The framework is scalable and contains nodes that run the necessary software in the cloud environment.

Lin et al. (2017) in F20 presented a mobile provable data possession framework that supports data dynamics operations via verification, outsourcing, block verification, and stateless verification by a trusted third party. A Hash tree structure and a Boneh-Lynn-Shacham (BLS) short signature were used in their scheme to support data dynamics operations. Data owners, CSPs, trusted third parties, and storage service providers are the four entities used in this framework.

Arvind & Manimegalai (2017), in F21, presented a secure data storage classification architecture (SDSCA) that protects data from illegal access. The framework consists of four components: 1) user agent, 2) mobile user, 3) CSP, and 4) broker. Data are captured from the mobile user are received by the user agent. The agent classifies this data into distinct levels, such as high, medium, and low. The data classification uses a superior naive technique made up of different algorithms (Naive-Bayes and K-Nearest Neighbor). This framework uses two-step methods to classify the data. The statistical features of the data are used to select the neighbouring data of the new data. When the data classification phase is completed, the Advanced Encryption Standard (AES) protects the data. The Homomorphic Encryption algorithm prevents the integrity of the data from being compromised. In the framework reported in F21, the security methods proposed allow the CSP and the mobile user to process the data without necessarily decrypting the data.

The authors in F22 presented a framework that uses a multimodal biometric system for authentication and access control to secure data in the mobile cloud environment. In this framework reported in F22, a fingertip image is obtained from the mobile user using the fingerprint sensors on their device. The image of the iris scan is captured using a high-definition image camera present in today's smartphones. This biometric information is stored in the cloud database and used to verify users in the mobile cloud environment. The framework (F22) provides an additional backup code that can be automatically generated and sent to the user's cloud profile for authentication with either biometric mean to enhance data security in the cloud server.

The authors of F23 proposed a framework that uses the steganography technique to enhance security in the MCC environment. The framework preserves embedded data unrevealed to the cloud, hence assuring the security of sensitive medical data when it's outsourced for encryption. The framework (F23) uses data preparation, outsourcing encryption, and data distribution to send classified medical images securely. In the data preparation stage, medical image detection is the salient object considered an important region of interest (ROI). The edge-directed data hiding method embeds the medical image into the host image. The selective encryption algorithm is used to encrypt the outsourced steganography image, and the result of the encryption is sent to the concerned user.

The authors in F24 presented a framework that adopts the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) for data security where data is encrypted. A lightweight data sharing scheme moves high-computationally intensive access from the MD to the cloud environment. The framework reduces user revocation costs by introducing property description fields to realize lazy revocation, a complex problem in program-based CP-ABE systems. A semi-trusted server enables cloud users to encrypt and decrypt data to reduce overhead in data processing using proxy encryption and decryption techniques.

The authors in F25 presented a framework that uses a trusted third party to provide a unique authentication for the mobile user for data access in the MCC environment. The trusted third party reduces identity theft risk since the user's identity information is contained in a secure identity management server (IDM). The IDM uses a homomorphic encryption and signature scheme to authenticate users on the cloud to grant access to their cloud resources. This framework provides privacy for the client's cloud service provider, but the risk of capturing the identity stolen through information interchange between mobile user and authenticator is not eliminated.

The framework reported in F26 presents a model to detect and isolate cyber-attacks that will severely impact the MCC environment using the deep learning technique. The authors of F26 claim that their model can achieve up to a 97.11% accuracy rate in detecting attacks in the MCC environment. The model proposed in F26 provides an attack detection module used to classify an incoming packet

request on the existing trained deep learning model. This framework trains the model in the offline mode used to detect malicious requests. The Attack module performs three basic duties: data collection and processing, attack recognition, and request processing. The deep learning model used in this framework has two phases: the learning process and feature analysis. The model uses the Principal Component Analysis (PCA) method for dimension reduction and the termination of optimal features. The learning phase of this model takes a long time and needs the pre-processing of data and features, which leads to an increase in the running cost. The pre-learning process of this model uses the Gaussian Binary Restricted Boltzmann Machine (GRBM) method to transform real values received from the input layer into binary codes used in the hidden layers.

The authors in F27 presented a framework that applies the concept of polymerase chain reaction and primer generation to ensure data confidentiality and integrity in the MCC environment. In this framework (F27), data is encoded in deoxyribonucleic acid (DNA) by converting the data into ASCII and binary form before converting it to the DNA format using the AGCT. The resultant DNA sequence is encrypted using symmetric encryption. This process is divided into different phases, including pre-processing data into numerical form, premier generation of keys from DNA sequences, the polymerase chain reaction encryption phase, and data integrity verification using the DNA Signature.

The authors in F28 presented a framework that uses a multi-layer intrusion detection technique to analyze incoming traffic and a decision-based approach to select a virtual machine. The scheme can best be described as a cognitive system consisting of a pattern matching engine and a knowledge base for profile-based traffic filtration analysis. The system employs unsupervised learning methods due to the multifaceted nature of incoming traffic and uses location information as a client's profile.

Agrawal & Tapaswi (2019) in F29 presented a framework that uses an agent-based multiple authority attribute-based encryption access control technique in the mobile cloud computing environment. The use of static and dynamic attributes in data encryption models is used to assure the security of data uploaded by the owners. To access data in the cloud, they must satisfy these attributes used in the encryption process. The proposed framework in F29 uses the mobile agent to deal with the loss of connectivity between the data owner and the cloud storage service provider. The framework reported in F29 also uses anonymous key issuing rules to manage encryption keys in their security framework.

The framework reported in F30 applies blockchain technology to tackle security and privacy issues in the MCC environment. The proposed framework focuses on a trust-based model to handle patient health records when information is exchanged by the MDs used by the patients and the healthcare provider's cloud storage and processing system. They implemented a prototype system as a proof of concept using Ethereum blockchain technology in real-time data sharing using Amazon CC services as their cloud storage provider. The performance evaluation carried out in their study was

reported to provide low network latency while maintaining a high level of security and privacy of user data during data exchange in the MCC environment.

The solution proposed in F31 enhances the quality of services MCC users enjoy by addressing undesirable delays MCC users encounter during user authentication and man in the middle attack during data transmission. They proposed a framework that provides a solution to the handover authentication issues in the MCC environment. The framework reported by the authors in F31 uses IEEE 802.11/LTE protocol to manage user's authentication while reducing the delay time.

The authors in F32 proposed a framework that tackles authentication issues in the MCC environment by applying a free pairing system to handle multi-server authentication problems faced by the MCC environment. The proposed framework eliminates the use of servers' storage for user verification by adopting a free pairing system that maintains user-oriented verifiers. They apply an elliptic curve cryptographical approach to secure user data during user authentication in the MCC environment.

The framework reported in F33 combat security issues associated with SMS-based authentication. The proposed framework applies offloading techniques to enhance the security of both the mobile and the cloud environment. The offload the two-factor authentication mechanism to the cloud and only grant access to a user if the authentication information sent by the user and the results of the cloud-based mutual authentication is valid. They use virtual smart card technology to authenticate each user request. The performance evaluation results reported by the authors show an improvement in tackling security issues regarding user authentication in the MCC environment.

The authors in F34 proposed a lightweight data sharing scheme that outsources high mobile device computational tasks to the cloud server. They used attribute-based encryption techniques to transform the data owner's plain text into ciphertext and upload the ciphertext to the cloud server. The framework reported in F34 outsources the key management tasks to a key generation centre third-party system. The third-party system manages both the private and public keys used for encryption and decryption. The framework allows the MCC users to obtain the decryption key without revealing it to another user using a secure channel. They reported that the performance evaluation of their proposed framework outperforms related works in literature.

The framework proposed in F35 uses a modular encryption approach to secure user data. The framework focuses on the security of patient health information by identifying and classifying each patient data's sensitivity and determining the required level of encryption needed to protect the data. The focus of the framework reported in F35 is on the security of user data at the cloud server to ensure that patient's health information is protected. They also provide a secured data sharing scheme between patients and medical doctors using the modular encryption standard. They reported

that their model competes favourably with the existing solution while maintaining a high standard of security of patient information.

2.5.2 SUMMARY OF THE REVIEW

The review frameworks can see a trend toward developing comprehensive frameworks (i.e., addressing multiple threats). In contrast, only one out of the eleven frameworks proposed between 2010 and 2014 managed more than five threat types, while in the period 2015 to 2021, the number was seven out of twenty-four. None of the frameworks addressed all the security threat types. All frameworks addressed threat type T1 (data breaches), which is concerned with preserving data confidentiality and privacy. Twenty-five frameworks also addressed threat type T4 (Insufficient Identity, Credential, Access, and Key Management an intrusion facilitator and thus, a precursor to threat T1). Nineteen of these frameworks were concerned with another data breach precursor, threat type T5 (account hijacking). The analysis of the review frameworks shows that frameworks proposed between 2015 and 2021 address more threats than those proposed between 2010 and 2014 (according to my findings, such frameworks can be categorized as more comprehensive).

The review analysis regarding the MCC security requirements addressed by each framework shows no specific trend regarding the number of fundamental MCC security requirements addressed. However, there was a slight increase in the number of frameworks addressing data integrity as an MCC security requirement, rising from one framework (F1) in the first half of the reviewed period to four frameworks (F20, F21, F27, and F30) in the second half of the review.

Furthermore, no frameworks addressed data confidentiality, integrity, and availability, which are the three basic security requirements of an information system (Table 2.5, column five). In addition, none of the frameworks addressed all seven fundamental security requirements. All frameworks addressed data confidentiality except one. Authentication focused on twenty-six frameworks, while five and eight frameworks addressed data integrity and availability. Most frameworks handle at least three security requirements, with data confidentiality and authentication being the most prevalent combination. Figures 2.2 and 2.3 show the graphical representation of the number of threats and security requirements addressed by all the thirty-five (35) frameworks used in this review. The analysis so far indicates that the frameworks that include IDS and apply ML approaches can offer more comprehensive MCC data security solutions.

Finally, the review demonstrates that various security approaches have been proposed to improve the security of user data in the MCC environment. Data protection approaches via encryption and access control via identity and trust management were the most prevalent. However, several frameworks focused on identifying threats to data confidentiality, such as those that occur within the "live" MCC environment, such as live monitoring and intrusion detection techniques. According to

the analysis of existing works carried in this study, shows that frameworks that incorporate IDS and utilize cognitive approaches can provide a more comprehensive MCC data security solutions. Additionally, due to the requirement to address security threats across the MCC environment's various layers, a comprehensive security solution may need to integrate and coordinate the use of multiple security controls such as stored data protection, access control, intelligent activities monitoring, and intrusion detection. This study adopted IDS techniques to create a more secure MCC data security environment and provide a more comprehensive solution to data security issues in MCC. This study reviews existing IDS frameworks in both mobile and cloud environments to understand this security technique in-depth, as discussed in the subsequent section.

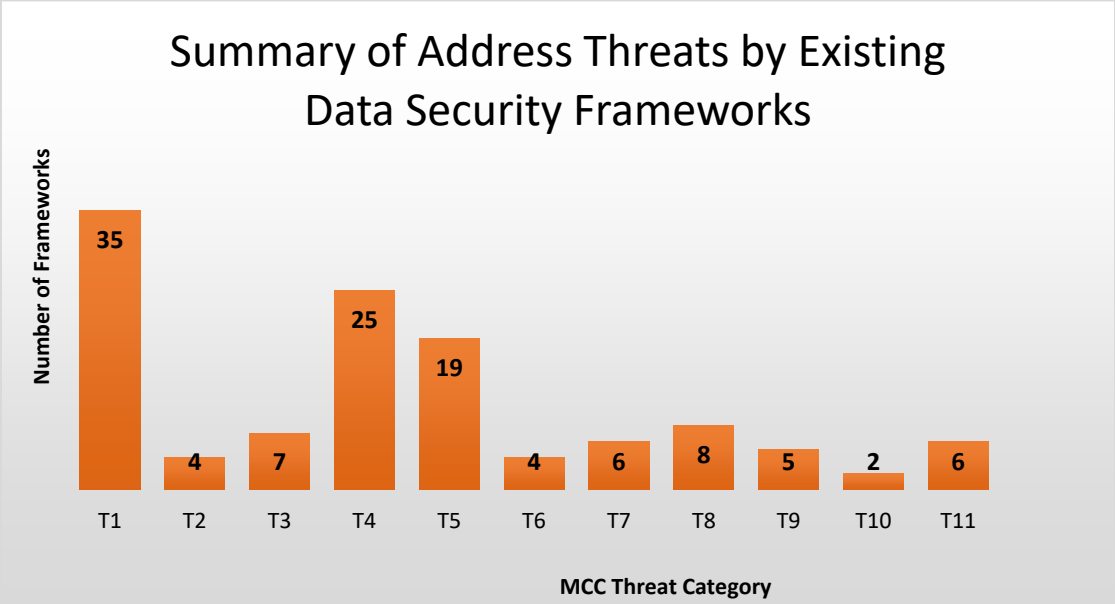

Figure 2.2. Summary of the Address Threats by Existing Frameworks

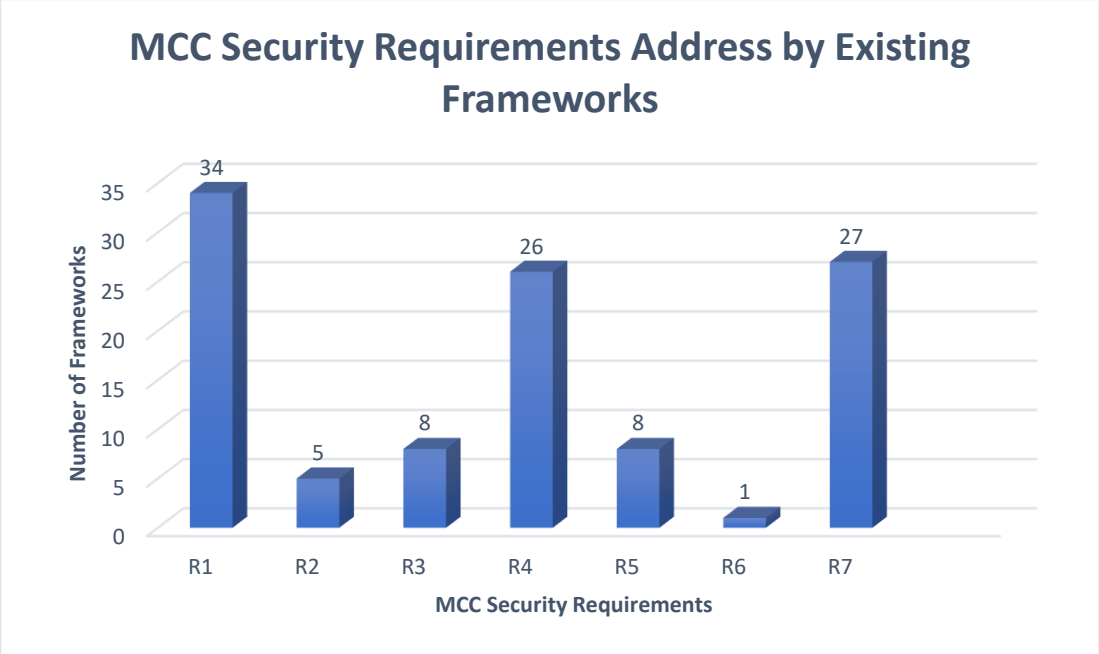

Figure 2.3. Summary of the Address Security Requirements by Existing Frameworks

2.6 Intrusion Detection System

Intrusion Detection System (IDS) monitor the activities that take place within the system or network to identify activities that violate the security policy of the systems. IDS is critical in cyber security because it enables a solid line of defence against cyber adversaries. The digital world has surpassed the physical world as the primary complement due to the widespread use of computer and network systems and their IoT services that efficiently execute users' tasks in a short time and at a low cost. Due to the rapid spread of information technology throughout the world, the need for securing network resources against cyber threats has grown. Since some existing technologies are not designed securely, it is critical to consider security by design when protecting them (Moustafa et al., 2019).

Each attacker employs unique and sophisticated techniques, posing severe threats to computer networks. When an attacker obtains essential information about a system, the system's confidentiality is violated, and when legitimate operations are interrupted, the system's availability and integrity are jeopardized. For instance, a denial of service (DoS) attack disrupts client systems, thereby violating the availability principle, whereas malware code hijacks the program's implementation, thereby violating the integrity principle (Pontarelli et al., 2012).

Patel et al (2013) stated that outsider attacks are those attacks that originate from an external source while Insider attacks occur when unauthorized internal users attempt to obtain and abuse privileged access. An IDPS is a software or hardware device that combines the functionality of an intrusion detection system to attempt to prevent potential incidents. IPSs can respond by attempting to thwart its success (Scarfone and Mell, 2007). The Intrusion Prevention System (IPS) modifies the attack's

content or modifies the environment's security. It can potentially alter the configuration of other security controls designed to thwart an attack, such as network reconfiguration of a device that prevents the attacker or the victim from gaining access to the system. IPS incorrectly classifies a legitimate non-intrusive surveillance system's normal activity as malicious and takes appropriate action in response to that detection.

2.6.1 Intrusion in The Mobile Devices

In MCC, intrusion can occur at the MD and in the cloud infrastructure. MDs have an architecture similar to personal computers (PCs), making them vulnerable to the same class of intrusions or malicious activities. Nowadays, MD users want to download apps for various purposes from app markets, including social networking, playing new games, and photography. In general, they are unconcerned about malicious apps and will download them regardless of whether they are infected with malware or not. Additionally, they will install and run these apps on their devices. As a result of these factors, the number of smartphones infected with malware and adware applications is rapidly increasing.

Kaspersky Lab reported the prevention of over 9 million malware, adware, and riskware attacks on MDs in 2021. Most of the detected attacks are associated with RiskTool apps. In addition, Kaspersky also reported the detection of 12,097 malicious apps that belong to mobile banking categories and 6,157 belonging to the mobile ransomware malware category. However, Android has some fundamental mechanisms for controlling app permissions, and the critical point is that a large number of unexpected (or unknown) attacks target smart gadgets. Users must employ a robust security solution to mitigate those attacks at the MD user layer in the MCC environment.

Mobile malicious application (MMA) is a type of hidden malware that operates in the background of the victim's device, completely undetected by the user. It can also execute or connect to other networks to obtain new instructions. Additionally, the MMA can manipulate the victim's device, resulting in specific results, such as abusing sensitive account specifications and information. For instance, an MMA can send a message to a particular phone number or reveal the user's location without the user's knowledge (Marforio et al., 2016).

In other words, the current version of MMAs is becoming increasingly sophisticated with malware that can run under the guise of legitimate apps, undetected by users and even anti-malware, and then perform some trick activities under the control of malicious users. The next generation of MMAs is expected to be even more intelligent, with botnet-like characteristics capable of controlling and hijacking victim devices (Karbab et al., 2016).

Malware is a type of malicious software that can steal information from users' devices, and, additionally, anti-malware programs can predict its behaviour. Numerous malicious apps provide

vulnerable entry points for malicious attacks on a user device in the MCC environment. The following are some of the categories of intrusions in MD (Inayat et al., 2017).

A. ***Virus***: A virus is a malicious app capable of imitating itself, and its various imitations can infect other applications, the boot sector, or files by attaching (or adding) themselves. The MD user must download and execute the malicious app to infect the device (La Polla et al., 2012).

B. **Spyware**: Spyware is malware that monitors the victim's device to monitor and control user activities such as location, contacts, calls, texting, and emailing. In some cases, it can transmit such data to another location via available networks (or email, SMS, etc.) and take control of a device without the user's knowledge (Inayat et al., 2017).

C. **Bot Process**: A mobile botnet is a collection of infected MDs remotely controlled by a botmaster (e.g., an individual who disrupts normal network traffic flow) without the users' knowledge. In other words, it creates a flaw within the intended app that allows attackers to gain complete control of the victim's device. It then begins communicating with it and receiving new instructions from specific servers. From a hacker's perspective, botnets are one of the most dangerous types of attacks because they can be used and controlled for various malicious purposes (most frequently, DDoS (Distributed Denial of Service) or spam attacks) (Alomari et al., 2012).

D. **Phishing apps:** A phishing app is a type of malware designed to look exactly like a legitimate application (for example, a mobile banking app, a market app, etc.) to steal sensitive information such as usernames, passwords, credit card specifications, and so on. Technically, these bogus apps impersonate legitimate apps on the victim's device by masquerading as trusted apps. Phishing apps can compromise the confidentiality of user input to hijack login authentication (Chaudhry et al, 2016). For example, a phishing app may mimic a mobile banking login screen to steal the user's account information (e.g., username and password). It is typically used to steal confidential information under the guise of fake mobile banking apps, which have become a recurring threat due to several reported incidents. According to Kaspersky Lab's malware analysis, a total of 12,097 mobile banking Trojans were discovered in 2021 that used phishing to steal users' account information (Kaspersky, 2021).

E. **Trojan**: A Trojan is a type of malware that allows unauthorized access to the victim's sensitive interactions, such as purchase transactions, premium rate calls, and so on. As a result, the goal of these malicious apps is to transmit under the guise of legitimate apps or file devices (La Polla et al., 2012).

F. **RootKit**: A rootkit is a hidden process that can run in the background of a victim's device and create malicious flaws for malware writers by infecting the operating system. In practice, this malware attempts to disable firewalls and anti-malware software or hide malicious user-space processes used to install Trojans (La Polla et al., 2012).

G. **Man-In-The-Middle Attack**: A Man-In-The-Middle (MITM) is a stealthy fraud that entails eavesdropping data transmissions between two communication devices. For example, the attacker

establishes a new connection between the target device and the server during a banking transaction. The hacker uses various techniques to split the direct connection into two new lines. The first connection is between the hacker and the server; the second is between the hacker and the victim's smartphone. This attack is one of the most effective threats since the TCP and HTTP protocols are based on the Unicode or ASCII standard. As a result, MITM attackers can decode and manipulate data streams as they pass through the target network.

2.6.2 INTRUSION IN THE CLOUD-INFRASTRUCTURE

CC environments are also vulnerable to intrusions that target the security of technology itself. Confidential information that users of cloud resources may store may become the target of an attack. Through obtaining unauthorized access, attackers may violate the privacy and confidentiality of cloud users' data stored in the cloud. Amongst others, the types of intrusion attacks prevalent in the CC infrastructure include insider attacks, flooding attacks, Denial of Service (DoS) attacks, user to root attacks, port scanning, Virtual Machine (VM) attacks, and covert-channel attacks. Such intrusion attacks are dangerous since they affect both the MD users and the CSP. Moreover, it is the responsibility of CSPs to provide adequate security protection of user information (Inayat et al., 2017). The following are types of intrusions in the CC environment.

A. **Insider attacks**: This type of attack allows a legitimate user of the cloud infrastructure to misuse unauthorized privileges by performing malicious activities in the cloud environment, thereby accessing, or modifying another user's information without authorization (Modi et al., 2013).

B. **Flooding attacks**: In this attack, intruders send many packets from an innocent host in the network, thereby making them not respond to legitimate traffic. In MCC, the user can access virtual machines when connected to the internet, which an attacker can use to cause DoS via the innocent host. Flooding attacks affect the availability of services to an authorized user when an intruder attacks servers that provide services to a user. It involves the availability of services offered by such servers in the MCC environment (Modi et al., 2013).

C. ***User to root attacks***: In this type of intrusion, attacker gains access to a legitimate user's account by sniffing passwords. As a result, they can exploit vulnerabilities to gain root-level access to the system. Buffer overflows, for example, are used to generate root shells from a process that is running as a root. It occurs when the application program's code exceeds the size of the static buffer. Frequently targeted are the mechanisms used to secure the authentication process. There are no universally accepted security mechanisms for preventing security risks such as insecure password recovery workflows, phishing attacks, and keyloggers. In the case of CC, the attacker gains access to valid user instances, allowing them to gain root-level access to VMs or hosts (Inayat, et al, 2017).

D. Port scanning: In this type of intrusion, the intruder attempts to locate and access an open port on the cloud network to launch attacks against cloud resources. This attack can reveal network-related information such as IP addresses, MAC addresses, routers, gateway filtering, and firewall rules. Some of the various port scanning techniques are CP scanning, UDP scanning, SYN scanning, FIN scanning, ACK scanning, and Window scanning. In a cloud scenario, an attacker can attack offered services by scanning for open ports where these services are provided (Modi et al., 2013).

E. Attacks on Virtual Machine (VM): In this type of intrusion, an attacker can gain control of installed VMs by compromising the lower layer hypervisor. Hackers may compromise the installed hypervisor and gain control of the host via these attacks. New vulnerabilities, including zero-day attacks, are discovered in VMs, attracting an attacker's attention, and allowing them to access the hypervisor or other installed VMs. Attackers use zero-day exploits before the target software's developer is aware of the vulnerability. A zero-day vulnerability in the Hyper VM virtualization application was exploited, resulting in the demise of numerous virtual server-based websites. (Inayat, et al, 2017).

F. Covert-channel attacks: In this type of attacks, the intruder exploits the weakness in the isolation of shared resources and use hidden part to steal confidential information. A passive attack enables the hacker to gain remote access to the infected node, thereby jeopardizing user confidentiality. Using backdoor channels, the hacker can take control of the victim's resources and turn them into zombies to launch a DDoS attack. Additionally, it reveals the victim's confidential information. As a result, the compromised system has difficulty performing routine tasks. In a cloud environment, an attacker can gain access to and control the resources of a cloud user via a backdoor channel and turn a VM into a zombie to launch a DoS/DDoS attack. A firewall (in the cloud) may be a standard solution for preventing some of the aforementioned attacks. Flooding and backdoor channel attacks can be detected using either signature-based or anomaly-based intrusion detection techniques (Wu et al., 2014).

2.6.3 TYPES OF INTRUSION DETECTION SYSTEM

There are four major types of intrusion detection systems (IDS) used in the CC environment: host-based intrusion detection systems (HIDS), network-based intrusion detection systems (NIDS), hypervisor-based intrusion detection systems, and distributed intrusion detection systems (DIDS).

A. HIDS: This type of IDS detect intrusion by analysing the information received from a host machine. The HIDS collects and analyses data from a specific host machine. HIDS monitors the machine for intrusions by collecting information such as the file system being used, network events, and system calls. HIDS monitors host kernel, file system, and program behaviour changes. When the system detects a deviation from expected behaviour, it alerts

the user to the presence of an attack. The effectiveness of HIDS is contingent upon the system characteristics being monitored. In the CC environment, HIDS is installed either on a host machine, VM, or hypervisor to monitor and analyze log files, security access control policies, and user login information to detect intrusive behaviour. If it's installed on a VM, the cloud user is responsible for its monitoring. Similarly, if installed on a hypervisor, the CSP is accountable for monitoring (Inayat, et al, 2017).

B. ***NIDS:*** This type of IDS detect intrusion by analysing the network packets to detect malicious activities in the network. NIDS can detect intruder by comparing the current network behaviour with previously observed behaviour (Modi et al.,2013). NIDS monitors network traffic to detect malicious activity such as denial-of-service attacks, port scans, and even attempted computer hacking. For intrusion detection, network data is compared to known attacks. NIDS employs a more robust detection mechanism to identify network intruders in real-time by comparing current behaviour to previously observed behaviour. NIDS is primarily concerned with monitoring individual packets' IP and transport layer headers to detect intrusion activity. NIDS employs intrusion detection techniques based on signatures and anomalies. NIDS has only a minimal view of the host machines. If network traffic is encrypted, the NIDS cannot decrypt it. Al-Hemairy et al. (2009) surveyed the security solutions used to detect ARP spoofing attacks. They concluded that the XArp 2 tool is an effective, commercially available security solution that accurately detects ARP spoofing attacks.

C. ***Hypervisor based IDS(Hy-IDS)***: This type of IDS allows users to monitor communication protocols among VMs and analyses the behaviour of these communications to detect possible intrusion (Modi et al.,2013). A hypervisor is a software platform that enables the execution of virtual machines. An IDS based on hypervisors is running at the hypervisor layer. It enables users to monitor and analyze communications between virtual machines (VMs), between hypervisors, and within hypervisor-based virtual networks. One of the advantages of hypervisor-based IDS is the availability of information.VM introspection-based IDS(VMI-IDS) is one type of hypervisor-based IDS.

Hypervisor-based IDS is a critical technique for detecting intrusions in virtual environments, particularly CC environments. VMI-IDS differs from traditional HIDS in that it directly observes the host's hardware states, events, and software states, providing a complete picture of the system than HIDS. The virtual machine monitor (VMM) manages hardware virtualization and provides isolation, monitoring, and interposition capabilities. VMI-IDS has access to the VMM so that code running in the monitored VM does not communicates with VMM via the VMM interface, enabling VMI-IDS to obtain VM state information, monitor specific events, and control VMs. This VMM interface comprises Unix sockets that allow you to send commands to and receive responses from VMM. Additionally, it enables physical memory access to the

monitored VM. The OS interface library translates VMM's low-level machine states into a higher-level OS structure. A policy engine is included for performing high-level queries on the monitored host's operating system. Even if the system is compromised, the policy engine responds appropriately (Modi et al.,2013).

D. DIDS: DIDS is a collection of IDS (for example, HIDS, NIDS, e.t.c.) distributed across a large network, all of which communicate with one another or with a central server that enables network monitoring. The intrusion detection components gather system information and format it for transmission to the central analyser. A foremost analyser is a computer that collects and analyses data from multiple IDS. The analysis employs a combination of anomaly and signature-based detection approaches. DIDS can be used to detect both known and unknown attacks because it combines the benefits of NIDS and HIDS (Modi et al.,2013).

2.6.4 INTRUSION DETECTION METHODS

In the CC environment, the following detection method are used in the design of cloud-based detection system: These methods are briefly described as follows:

A. Signature-based IDS: This detection method tries to define a set of rules or signatures used to predict and detect the consequently known patterns of an attack. This method usually achieves high accuracy in detection with minimal false positives in recognizing intrusions in a specific environment. This method is used to identify a known attack in a cloud-based system. The positioning of the IDS in the cloud network is significant and determines the categories of external or internal attacks it can effectively detect. If an IDS that uses the signature-based method is placed at the front end of the cloud network, it can detect known external attacks but not detect internal intrusion, but if placed at the back end, it can detect both internal and external attacks (Patel et al., 2013).

B. Anomaly-based IDS: This detection method is concerned with identifying a malicious attack that seems anomalous concerning normal behaviour in a cloud network. Different techniques are used in this type of detection, such as data mining, statistical modelling, and ML. This approach involves collecting data relating to the behaviour of authorized users over some time and then applying statistical tests to the preserved behaviour to determine whether that behaviour is legal or not. It has the advantage of identifying attacks that have not been found previously. This method applies to the cloud-based infrastructure for detecting unknown attacks at different levels. Modi et al. (2013) reported that the adoption of soft computing techniques is used to improve the accuracy and efficiency of intrusion detection using anomaly-based detection methods. Some of these soft computing techniques used in anomaly-based intrusion detection are Artificial Neural Network (ANN), Fuzzy logic, Association rule mining, Support Vector Machine (SVM), Genetic Algorithm (GA), and so on.

C. ***Hybrid IDS***: In these IDS, the capabilities of an existing IDS are enhanced by combining the two methods (signature-based and anomaly-based) to detect both known attacks and unknown attacks (Patel et al., 2013).

2.7 Analysis of Current IDS Frameworks in CC, MD and MCC Environment

Following up on the summary presented in section 2.6.2, this analysis of the current IDS frameworks in the CC, MD, and MCC environments addresses the second research objective stated in chapter one, section 1.3. The search for the articles across the different databases (IEEE, Science Direct, ACM, and Springer Link) resulted in 412 articles published between 2010 and 2020. The search string used the Boolean OR and AND to construct the keyword expression used in the search: (intrusion AND detection AND ((Mobile Cloud) OR MCC)) OR (anomaly AND detection AND ((Mobile Cloud) OR MCC)) OR (signature AND detection AND ((Mobile Cloud) OR MCC)) OR (internal AND attack AND ((Mobile Cloud) OR MCC)).

This study selected sixty-five peer-reviewed articles for journals and conferences written in English. The selected articles proposed an IDS framework targeting the CC, MD, or MCC environment. The selection of this relevant articles outside the MCC domain was limited since only a few works that use IDS have been proposed in the MCC environment. Analysing the existing IDS frameworks that target either the CC, MD, or MCC environment reveals possible research gaps and addresses research objectives 2 in chapter one, section 1.3. The articles were analysed (Table 2.6) using the following predefined set of dimensions:

IDS Type(D7): This dimension specifies the type of IDS proposed in a specific framework (for example, HIDS, NIDS, DIDS, or Hy-IDS).

Detection Method(D8): This dimension identifies the method of intrusion detection used in the proposed framework (for example, Signature-based (SB), Anomaly-based (AB) and Hybrid (HB)).

Target Environment(D9): This dimension specifies the environment the framework was built to work in, such as MD, CC, or MCC.

Machine Learning (ML) Component(D10): This dimension indicates whether or not the framework incorporates any ML process or algorithm for intrusion detection.

Prevention Component(D11): This dimension indicates whether or not the framework provides an intrusion prevention mechanism in addition to the intrusion detection ones.

Table 2.6. Analysis of IDS Frameworks

ID.	Source	D7	D8	D9	D10	D11
F36	Dhage et al., 2011	DIDS	AB	CC	Yes	No
F37	Houmansadr et al., 2011	HIDS	SB	CC	No	Yes
F38	Ulltveit-Moe et al., 2011	HIDS	AB	MD	No	No
F39	Modi et al., 2012	NIDS	HB	CC	Yes	No
F40	Khune & Thangakumar, 2012	HIDS	SB	CC	No	Yes
F41	Yan, 2012	NIDS	AB	CC	Yes	No
F42	Yassin et al., 2012	NIDS	SB	CC	No	No
F43	Ficco et al., 2012	DIDS	SB	CC	No	No
F44	Man & Huh, 2012	NIDS	SB	CC	No	No
F45	Patel et al, 2012	NIDS	HB	CC	Yes	Yes
F46	Roshandel et al., 2013	DIDS	HB	MD	Yes	Yes
F47	Dolgikh et al., 2013	NIDS	AB	CC	No	No
F48	Yazji et al., 2014	HIDS	AB	MD	Yes	Yes
F49	Milosevic et al., 2014	HIDS	AB	MD	Yes	No
F50	Li et al., 2014	NIDS	AB	MD	Yes	No
F51	Idrees & Muttukrishnan, 2014	NIDS	AB	CC	Yes	No
F52	Moorthy & Masillamani, 2014	DIDS	HB	CC	Yes	No
F53	Pandian & Kumar, 2014	NIDS	AB	CC	Yes	Yes
F54	Qi et al., 2014	NIDS	AB	MD	Yes	No
F55	Kumar & Hanumanthappa, 2015	NIDS	SB	CC	No	No
F56	Marengereke & Sornalakshmi, 2015	NIDS	SB	CC	No	Yes
F57	Shi et al., 2015	DIDS	AB	MCC	No	Yes
F58	Mehmood et al., 2015	Hy-IDS	SB	CC	No	No
F59	Toumi et al., 2015	DIDS	AB	CC	Yes	No
F60	Fischer et al., 2015	Hy-IDS	AB	CC	Yes	No
F61	Modi, 2015	NIDS	HB	CC	Yes	Yes
F62	Singh et al, 2016	NIDS	AB	CC	No	No
F63	Hou et al., 2016	HIDS	AB	MD	Yes	No
F64	Hatcher et al., 2016	HIDS	AB	MD	Yes	No
F65	Dbouk et al., 2016	NIDS	SB	CC	No	No
F66	Kholidy et al., 2016	DIDS	HB	CC	No	Yes
F67	Pandeeswari & Kumar, 2016	Hy-IDS	AB	CC	Yes	No
F68	Nagar et al., 2017	DIDS	HB	CC	No	Yes
F69	Tong & Yan, 2017	HIDS	HB	MD	No	No
F70	Nezarat, 2017	Hy-IDS	AB	CC	No	No
F71	Moloja & Mpekoa, 2017	NIDS	SB	CC	No	Yes
F72	Balamurugan & Saravanan, 2017	NIDS	AB	CC	No	Yes
F73	Idrissi et al., 2017	DIDS	SB	CC	No	Yes
F74	Nezarat & Shams, 2017	Hy-IDS	AB	CC	No	No
F75	Raja & Ramaiah, 2017	NIDS	AB	CC	Yes	No
F76	Velliangiri & Premalatha, 2017	NIDS	AB	CC	Yes	No
F77	Sohal et al., 2018	NIDS	AB	CC	Yes	Yes
F78	Li et al, 2018	NIDS	AB	CC	No	Yes
F79	Ghribi et al., 2018	NIDS	AB	CC	No	No
F80	Nguyen et al., 2018	NIDS	AB	MCC	Yes	No

ID.	Source	D7	D8	D9	D10	D11
F81	Ravji & Ali, 2018	NIDS	HB	CC	No	Yes
F82	Qin et al., 2018	HIDS	AB	CC	Yes	No
F83	Achbarou et al., 2018	DIDS	HB	CC	No	No
F84	Besharati et al., 2018	HIDS	AB	CC	Yes	No
F85	Kim et al., 2018	NIDS	AB	CC	Yes	No
F86	Modi & Patel, 2018	NIDS	HB	CC	Yes	No
F87	Peng et al., 2018	NIDS	AB	MCC	Yes	No
F88	Rajendran et al., 2018	NIDS	AB	CC	Yes	No
F89	Ribeiro et al., 2018	HIDS	AB	MD	Yes	No
F90	Dey et al., 2019	DIDS	AB	MCC	Yes	No
F91	Weng & Liu, 2019	NIDS	AB	CC	Yes	No
F92	Ribeiro et al., 2019	HIDS	AB	MD	Yes	No
F93	Zhou et al., 2019	HIDS	AB	MD	Yes	No
F94	Mugabo & Zhang, 2020	NIDS	AB	MCC	Yes	No
F95	Kim et al.,2020	NIDS	AB	MD	Yes	No
F96	Lima et al.,2020	HIDS	AB	MD	Yes	No
F97	Barbhuiya et al., 2020	NIDS	AB	MD	Yes	No
F98	Manikanthan et al., 2020	HIDS	AB	MD	Yes	No
F99	Subramaniam, 2020	DIDS	SB	MCC	No	No
F100	Gaharwar & Gupta,2020	NIDS	AB	MD	No	Yes
Note: D7-IDS Type, D8-Detection Method, D9-Target Environment, D10-ML Component, D11-IPS Component						

2.7.1 REVIEW OF IDS FRAMEWORKS THAT TARGETS CC INFRASTRUCTURE

This review shows that 65% of the selected frameworks target IDS implementation in the CC environment. The frameworks proposed in F37, F40, F82, and F84 are HIDS types that target the CC environment. The frameworks presented by the authors in F37 and F40 use an SB detection approach and proxy servers for in-depth forensic analysis of files stored locally on the device for intrusion detection in the CC environment. The proposed frameworks reported by the authors in F82 and F84 use the AB detection approach with ML techniques. F82 uses a mobile agent to collect data from each host for its detection process automatically. However, F84 focuses on protecting the VMs in the CC environment. In this review, the NIDS proposed solution that uses the SB detection method has the detection engine located on the cloud server. The analysis of the reviews shows that only two frameworks that use the SB detection method proposed by the authors in F56 and F71 incorporate the IPS component to mitigate malicious activities detected in the CC environment. The framework presented in F44 uses correlated alerts for its detection process. Some solutions that target the CC environments shift the detection engine to the MD node, as reported by the authors in F55 and F65. The solution presented in F42 enhances the SB detection method by automatically updating a new signature.

The NIDS frameworks that use the AB detection method for analysing network traffic apply various ML techniques to detect intrusions in the CC environment. F76 and F88 were concerned with detecting DoS attacks in the CC infrastructure. Framework F91 provides a novel approach for

anomaly detection using statistical time series features. F85 applies both supervised and unsupervised ML techniques to improve the detection and classification of attacks in the CC environment. The security approach presented in F78 incorporated mobile immune agents, while F79 uses the correlation of alerts to detect malicious activities in a network. The frameworks presented in F62, F75, and F77 use ML techniques. The solution in F77 was concerned with predicting the malicious devices in the cloud network. At the same time, F62 relies on identifying network paths where disruption occurs as a result of longer transmission times and reduced speed in transmission for intrusion detection.

The NIDS solutions presented in F72 and F53 address security issues concerning communication using cloudlet controllers and virtual private networks (VPNs), respectively. However, the frameworks presented in F47, F41, and F51 analyze system calls and model the device's behaviour to enable the IDS to identify attacks. The NIDS frameworks presented in F39, F45, F61, F81, and F86 apply the HB detection method to identify intrusions in the CC environment. Most of these solutions combine SNORT with ML algorithms. However, the framework proposed by the authors of F81 uses honeypot technology to produce an early warning about possible threats and attacks.

The frameworks presented by the authors in F58, F60, F67, F70, and F74 are Hy-IDS. Only F58 uses the SB detection method, while F60, F67, F70, and F74 use the AB detection method in their detection engines located at the CC infrastructure. However, none of the Hy-IDS uses a hybrid detection method. Mobile agents are common in these frameworks reported by the various authors mentioned above. These mobile agents carry intrusion alerts from each virtual machine in the cloud to a management server for analysis to detect distributed intrusions at the hypervisor layer. The solutions presented in F73, F43, F36, F59, F52, F66, F68 and F83 are DIDS. The authors in F43 and F73 use the SB detection method to design their proposed frameworks. In addition, the HB detection method was applied to F52, F66, F68, and F83. The framework reported in F36 and F59 uses the AB detection method for managing intrusions in the CC environment.

2.7.2 Review of IDS Frameworks that Targets MD Infrastructure

The frameworks proposed by the authors in F38, F48, F49, F63, F64, F69, F89, F92, F93, F96, and F98 are HIDS types that target the MD environment. The detection engine for these frameworks was located at the device level, except for F48 and F49. The framework reported by the authors in F48 focuses on device resource optimization and places its detection engine in the cloud, while F49 places its detection engine on the device. HB detection was applied in the framework presented in F69, while the other frameworks adopted the AB detection method. However, none of the HIDS types targeting the MD environment uses the SB detection method. The dynamic and static analysis of malicious apps in the MD node using system calls was used in the proposed framework in F69. Similarly, the framework proposed in F63 extracts system calls from the applications that reside on

the devices and constructs a weighted direct graph. It applies a deep learning algorithm to detect new attacks. Using ML techniques, the framework proposed by the authors in F89 presents an autonomous detection of malicious activities (known and unknown attacks).

The frameworks presented in F38 and F48 use location-based services to detect intrusions at the MD node. F49 runs a local malware detection algorithm at the MD node to check for a known malware family. The security solution presented in F93 and F92 analyses system calls and system log files respectively to determine if a given app is malicious or not. The security techniques in F64 use the Google Cloud Messaging service in malware detection. The NIDS frameworks targeting the MD environment in this review are seen in F50, F54, F95, F97, and F100. The NIDS based frameworks that target the detection of malicious activities at the MD node use the AB detection method. The detection engine in F50 resides in the cloud, while F54, F95, FF97, and F100 reside on the device. The framework presented in F46 uses the DIDS and the HB detection approach. The detection engine in F46 resides on both the device and the cloud.

2.7.3 Review of IDS Frameworks that Targets MCC Infrastructure

The authors of F57, F80, F87, and F90 proposed frameworks that target the MCC environment. F57, F90, and F99 are DIDS types, while F80, F87, and F94 are NIDS types. All the frameworks that target the MCC environment presented in this review apply the AB detection approach with ML techniques, except for the framework reported by the authors in F99, which uses the SB detection method. F57 has the only prevention module. In the framework presented in F80, the attack detection module analyses incoming requests and classifies each request as normal or suspicious based on the trained deep learning model. The security techniques used in F87 apply balanced iterative reducing and clustering using hierarchies with principal component analysis for simulation experiments to demonstrate intrusion in the MCC environment. The framework in F90 presents an ML-based IDS that secures the data collected and data fusion in a distributed environment. This framework (F90) uses a multi-layer intrusion detection technique to analyze incoming traffic and a decision-based technique to select a virtual machine.

2.7.4 Summary of the Review

The analysis of articles shows that most of the proposed IDS frameworks target the CC environment, as shown in Figure 2.4. The result of the analysis presented in Figure 2.4 shows that the MCC environment has the least framework. The analysis of the reviews shows that 9% of the selected IDS frameworks target the MCC environment, while 26% and 65% of the selected IDS frameworks target the MD and CC environments, respectively.

As shown in the preceding subsections, most of the proposed IDS frameworks do not provide a prevention technique once an intrusion is detected. The results in Figure 2.5 show that, out of forty-two frameworks that target the CC environment, only fourteen frameworks proposed a prevention

technique to handle intrusions detected in this environment. Similarly, only one (F57) out of six frameworks proposed a prevention technique in the MCC environment. In addition, three (F46, F48, and F100) out of seventeen frameworks have a prevention module in their detection engines that targets the MD environment. It was observed that some of the frameworks that have a prevention feature only deploy a passive prevention method, which requires the user or administrator to take appropriate action to mitigate suspected attacks. This is a challenge that needs to be addressed. A good prevention technique that requires a fast response to intrusion without the user or administrator taking any necessary action will help to enhance security in this environment.

The review of existing IDS shows that only a few frameworks incorporate ML in the detection of intrusion, as shown in Figure 2.5. The use of a hybrid ML technique, which includes a supervised, semi-supervised, or unsupervised learning model, will help to improve security in this environment. The result of the analysis of existing IDS solutions in the review shows that the majority of the existing IDS frameworks use the AB detection method. The AB detection method is associated with a high rate of false alarms and required a lot of training time and computational resources. The HB detection method is reported to be better because it has a higher degree of accuracy in its detection process since it combines both the SB and AB detection methods. A good security framework that produces better accuracy in detection and low false alarms is required to enhance the security of the MCC environment.

Several dimensions adopted in this study to analysed existing data security frameworks and existing IDS and IDPS frameworks(F1-F100) and models together with the validation metrics used in the evaluation of existing IDPS model proposed in extant literature are summarized in Table 2.7 and Table 2.8, respectively.

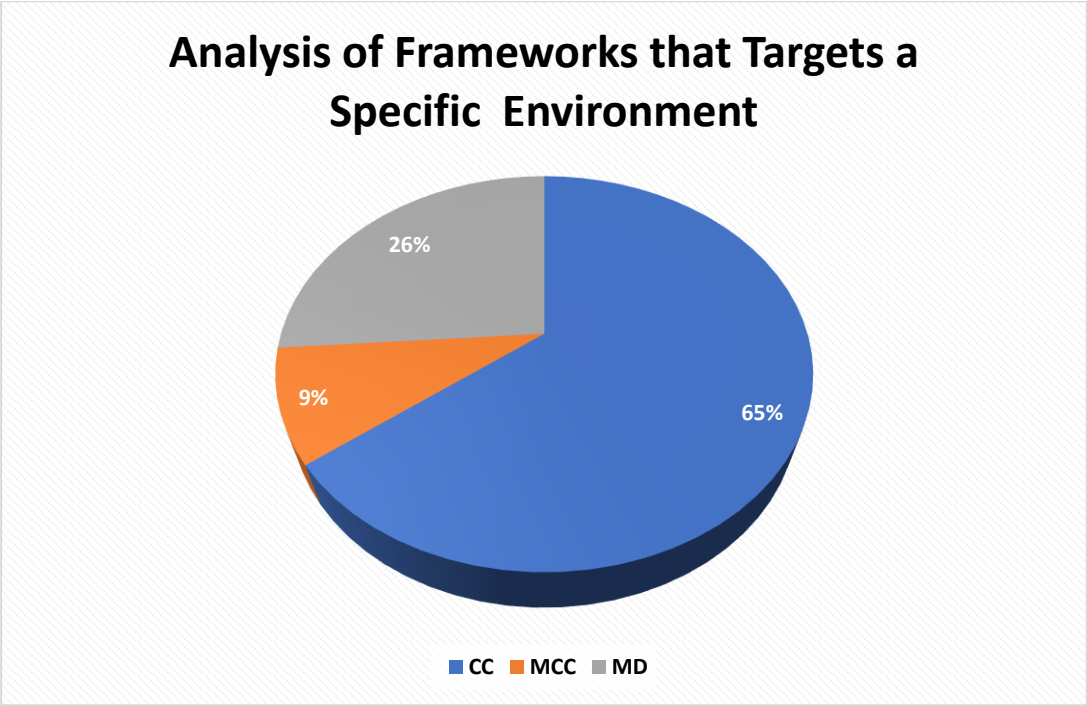

Figure 2.4. Analysis of IDS Frameworks and their Target Environment

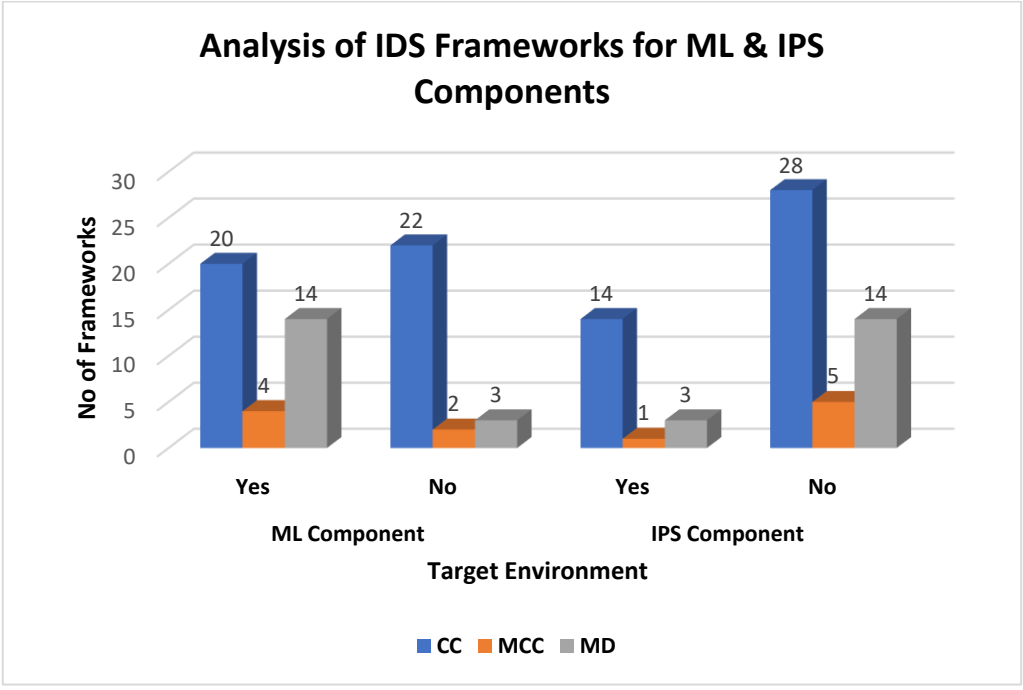

Figure 2.5. Analysis of IDS Frameworks for ML & IPS Components

Table 2.7. Dimensions Adopted in this Study for Framework Analysis

ID	Dimension Name	Description
D1	Domain	This dimension describes the proposed security framework's architectural layers (MCUL, MNCL, and MCSPL).
D2	Threats	This dimension describes the threats that the existing data security framework in literature addresses using the eleven threats (T1 to T11) presented in Table 2.3.
D3	Security Requirements	This dimension describes the security requirements that the existing data security framework in literature addresses, as presented in Table 2.1.
D4	Security Approach	This dimension describes the proposed framework's techniques to protect user data (such as encryption, biometrics, access control, activity monitoring, and intrusion detection).
D5	Cloud Level Trust	This dimension identifies the MCC components that are trusted by the framework. This study assumes that using a trusted third party within a proposed framework shows that the cloud infrastructure is not trustworthy. Similarly, if there is no third-party component in the proposed framework, the cloud infrastructure is assumed to be trusted.
D6	Comprehensiveness	This dimension is used to measure the number of threats a framework addresses. A framework is considered more comprehensive in this study if it addresses five or more threats; otherwise, it is less comprehensive.
D7	IDS Type	This dimension specifies the type of IDS proposed in a specific framework (for example, HIDS, NIDS, DIDS, or Hy-IDS).
D8	Detection Method	This dimension identifies the method of intrusion detection used in the proposed framework (for example, Signature-based (SB), Anomaly-based (AB) and Hybrid (HB)).
D9	Target Environment	This dimension specifies the environment the framework was built to work in, such as MD, CC, or MCC.
D10	Machine Learning (ML) Component	This dimension indicates whether or not the framework incorporates any ML process or algorithm for intrusion detection.
D11	Prevention Component	This dimension indicates whether or not the framework provides an intrusion prevention mechanism in addition to the intrusion detection ones

Table 2.8 Existing Validation Metrics Used for the Evaluation of Existing IDS and IDPS

S/N	Metrics	Description	Calculating Function
1	Classification Accuracy (CA)	This is the total percentage of the correctly classified malicious and benign apps in any given sample dataset.	$CA = \frac{TP+TN}{TP+FP+TN+FN} \times 100$
2	Error Rate (ER)	This is the total percentage of all wrongly classified benign and malicious apps in the entire dataset.	$ER = \frac{FP+FN}{TP+FP+TN+FN} \times 100$
3	Precision Rate (PR)	This is the total percentage of correctly classified results of all malicious apps that belongs to the benign labelled in the dataset	$PR = \frac{TP}{TP+FP} \times 100$
4	Recall Rate (RC)	This is the total percentage of malicious apps that are correctly predicted as malicious apps in the dataset.	$RC = \frac{TP}{TP+FN} \times 100$
5	False Positive Rate (FPR)	This is the total percentage ratio of malicious apps classified wrongly to the actual numbers of the malicious samples in the dataset	$FPR = \frac{FP}{FP+TN} \times 100$
6	False Negative Rate (FNR)	This is the total percentage ratio of benign apps classified wrongly to the actual number of samples in the dataset.	$FNR = \frac{FN}{FN+TP} \times 100$
7	False Alarm Rate (FAR)	This is the total percentage average ration of malicious and benign apps that are misclassified.	$FAR = \frac{FPR+FNR}{2}$
8	F-Measure (FM)	This is the harmonic mean of the proposed classifier which is obtainable from the value of both PR and RC	$FM = 2 \times \frac{PR \times RC}{PR+RC}$
Note	True Positive (TP)	The total number of malicious apps that were classified correctly	
	True Negative (TN)	The total number of benign apps that were classified correctly	
	False Negative (FN)	The total number of malicious apps that were incorrectly classified as benign apps.	
	False Positive (FP)	The total numbers of benign apps that were incorrectly classified as malicious apps.	

2.8 STATE OF SECURITY FOR MOBILE DEVICES IN THE MCC ENVIRONMENT

Attacks on the MD layer (MCUL) of the MCC infrastructure have increased in recent times due to the technique adopted by hackers. These attacks are different from those on desktops or enterprise information systems that are application-specific attacks such as port scanning or SQL injection attacks. The recent attacks on the MCUL in the MCC environment are in the form of users installing apps on their devices that require more than the usual number of permissions in order to perform malicious activities without user consent. Most of the user-installed apps in the MCC environment utilize more permissions, which enables the apps to retrieve privacy-related data that can be exploited by hackers for financial gain. In addition, the process of injecting malicious codes into legitimate apps by hackers has raised security concerns about the safety of user data stored locally

on both the device and the cloud infrastructure in the MCC environment. Such apps in the MCUL can serve as a point of entry for possible intrusion into the cloud services in the MCC environment.

The popularity of Android devices amongst MCC users has motivated malware developers to target Android-based devices in this environment. One of the major contributing factors explaining why this platform has been targeted so easily is the open application publication policy, which allows its users to install apps from both official and non-official market stores. The relatively low cost of acquiring Android devices has been one of the reasons why the number of users of the platform has increased significantly. These devices are part of our daily lives as we use them to access sensitive resources that are business or work specifically. The increased reliance on these devices for our daily activities such as financial transactions, shopping, communication, and storing of private data has made them a prime target for malware developers. The recent trend by malware developers targeting mobile users, especially those on the Android platforms, has raised concerns with regards to the security of the user layer of the MCC environment.

The end-users of the MC technology are exposed to risk through a number of different vulnerabilities. Different techniques have been proposed to solve the problem of MD users downloading malware mobile apps, focusing on mobile app security characteristics such as permissions, intent, API calls, system calls, kernel operation, and resource usage. For example,

Alazab *et al.* (2020) proposed a malware detection model that uses permission and API calls. They proposed three different strategies for selecting relevant API calls that will improve the chances of detecting a malicious app. Using statistical and ML approaches, Idrees *et al.* (2017) proposed a novel malware detection system that uses a combination of permissions and intent to identify malicious apps. The model reported in their work was evaluated using 1,745 Android apps, and the authors reported a classification accuracy of 99.80%. However, the model was not implemented in a real-life Android device. The authors only reported their experimental work conducted on their personal computers.

Hou *et al.* (2016) proposed a framework named Deep4MalDroid that could detect malicious apps using deep learning techniques. The framework proposed in their work applies the concept of weight graph for feature extraction of system calls. They propose a novel approach that uses dynamic analysis called component traversal to execute code routines in each given Android application to extract the Linux kernel system calls of the apps running on the devices. The model reported by these authors uses a dataset of 3,000 Android apps for both training and testing of their proposed system.

Qi *et al.* (2014) proposed a network behaviour-based malware detection for Android devices, which is made up of a network behaviour monitoring module, a network behaviour analysing module, and a storage module. The monitoring module is used for extracting features of network behaviour. A feature vector of the network behaviour characteristic was constructed based on the impact of malware on Android devices. The vector contains the process ID, the start and end time of the network connection, up/downflow, source/destination IP address, protocol type, and source/destination port number. They evaluate their model using 1260 malware apps.

The security model proposed by Hatcher et al. (2016) detects malicious activities in Android OS mobile devices in which the detection process of these activities utilizes both static and dynamic analysis approaches, simultaneously providing rapid and intuitive security with predictive capabilities. The system is centred around four primary components, the Android App, the Security Server, Google Cloud Messaging (GCM) service, and the Analysis Module. The authors reported using 241 malware and 241 benign apps for the detection based on permissions in their experiment. Similarly, the authors use 91 malware and 95 benign apps for the detection based on system call data.

Ribeiro et al. (2020) proposed a system that dynamically analyses device behaviour by monitoring deviations in device behaviour characteristics of certain features such as total CPU usage, memory consumption, total outgoing and incoming network traffic, battery usage, and so on using machine learning and statistical algorithms to classify it as either benign or malicious.

Zhou et al. (2019) proposed a model for malicious attack detection using dynamic features extraction of mobile apps constrained in 166 dimensions and applied a novel ML classifier to identify malware apps by triggering an alarm. In their experimental work, system calls from both benign and malicious Android-based apps were collected. The detection engine reported in their proposed model acquires runtime system calls from unknown Android apps and forms the feature vector table, which is fed to the detection model to evaluate the behaviour of the unknown app. The behavioural analysis of the system calls enables their model to identify apps that are performing malicious activities on the user device.

Feizollah et al. (2017) proposed a model named Androdialysis. The proposed system in their work uses Android intent as a feature for identifying malicious apps. The experimental work carried out by the authors used both permissions and intent to detect malicious apps. They achieve better accuracy using intent as compared to permission. They use a total of 7,406 apps, with an overall detection rate of 91% using intent and 83% using permissions.

Arp et al. (2014) proposed a lightweight approach (DREBIN) for detecting malicious apps using permissions, hardware components, restricted API calls, and network addresses. The proposed system was evaluated using 123,453 benign apps and 5,560 malware apps. The authors reported an overall classification accuracy of about 94%. Similarly, Saracino et al. (2016) presented a model called MADAM. MADAM analyses apps at four distinct levels: kernel, application, user, and package to detect malicious activities in apps. They tested their model with 2,800 apps and got an overall classification accuracy of 96%.

Li et al. (2018) presented a model called SigPID, which uses permission to identify malicious apps. These models apply the concept of selecting significant permissions that are relevant to distinguishing malicious apps from benign apps. They identify 22 permissions that are significant in identifying a malicious app. The overall classification accuracy reported in their work is 93.62%.

2.9 Mobile Application Risk Factors and Assessment

Today's mobile internet is so popular that using mobile apps has become a crucial component of our daily lives. These mobile apps offer fantastic features like file sharing, email services, and entertainment, among others. Due to the extensive permissions that modern apps require, there is a significant risk that they could gain confidential data from MD users in the MCC infrastructure. Users in the MCC environment download applications via app stores like the Apple App Store and the Google Play Store (Feng et al., 2019).

These apps are installed on their devices with permissions provided to allow access to their contacts, photo gallery, and location data, etc. While some of the apps may not be malicious, they can undermine user privacy by giving others access to personal information and sensitive data. The Internet of Things (IoT), a developing technology that enables everything in our daily lives to be connected, relies heavily on MCC devices like smartphones as end user interface. However, exchanging data between devices poses potential security risks. Due to a malicious programme that lives on a specific connected device, there is a chance that linked devices in the IoT environment could also become infected. These dangers are related to the vulnerability of mobile apps, particularly for users of the Android mobile operating system (Kim et al, 2020).

The goal of the risk assessment is to offer a quantitative estimate of the likelihood that a resource accessed by an app would harm users. Particularly in the MCC environment, the risk assessment of mobile apps has not drawn much attention. However, as discussed in the preceding section, the majority of related research efforts focus on the capacity to distinguish between malicious and benign apps in the mobile environment. For example, the work reported in Feng et al. (2019) uses app permissions and descriptions to determine the risk level of an app. Similarly, Wang et al. (2013) proposed a framework that uses apps permission and quantitative analysis to determine the riskiness of an app.

A risk assessment approach called XDroid was reported by Rashidi et al. (2019) to monitor the resource utilisation of Android devices based on the permissions requested. For the purpose of providing an adaptive risk assessment of the apps that are installed on the device, they adopted the use of the hidden Markov model and an online learning technique. Users of the model described in their work can create a unique profile of the resources they want to track. Based on the user-approved device resources specified by the user, the model proposed by Rashidi et al. (2019) utilises a probabilistic method to model app behaviour and inform the user of suspicious actions. However, their work suffers from relying solely on the user to decide which resources should be monitored without considering the user's capability to recognise the proper resources to be watched.

A model called PUREDroid was introduced by Alshehri et al. (2019) that assesses the security risk of Android apps based on the user's consented authorization. The model put forth by Alshehri et al. (2019) calculates the extent of the harm that could result from users of Android devices granting too many permissions. The requested and non-requested permissions of each mobile apps are represented using two orthonormal states vectors by PUREDroid, which calculates the risk score of each app based on the app category. According to PUREDroid, each app's risk score is calculated based on how frequently both good apps and malicious apps have asked for this permission. However, their algorithm ignores the fact that some malicious apps request a disproportionate number of permissions and only assigns a high-risk score to the dangerous apps. However, they do not use any kind of ML techniques in their work to enhance the performance of their model.

A model named RISKMON was proposed by Jing et al (2014) to assist users in understanding and reducing security risks related to mobile apps, particularly in the Android mobile environment. The approach adopted by their model combines user expectations with runtime behaviour of trusted apps to create a baseline risk score. The risk score baseline results are applied to model an app's real-world behaviour. When an app tries to access sensitive or important device resources, RISKMON assigns a score, and based on the risk baseline score, generates cumulative scores. By assuming that user assets can only be accessed with protected permission, the research presented in RISKMON analyses permission-protected only systems. They did, however, propose an automated permission revocation without considering the user's agreement, which might have an impact on the user's actions while utilising certain of the services that an app requests.

In addition, the work reported by Son et al (2021) applies app static analysis approach to assess the risk of apps that resides on user devices. The work of Son et al (2021) focus on how each mobile app requests personal data and how the pattern of each request. The model proposed in their work, analyse personal data that most apps with the same commercial objective (such as social networks, commerce, sports, etc.) demand in order to define the "regular behaviour." Following that, they

determine how much the target app's signature deviates from the typical access pattern of the associated set of apps to estimate the risk of an app.

In summary, several works in literature have used different risk assessment models to determine the risk poses by different apps in the mobile environment. Some of the existing work in the literature adopted the permission-based model, app description-based model, user review-based model and API-based model to determine the risk of an app running on the user device. Nonetheless, majority of the work in literature have failed to clearly state the risk factors associated with their proposed model. However, the work reported by Sharma and Gupta (2018) defines the risk factors that can used in the design of a risk assessment model. The proposed prototype system implemented in this study adopted the risk factors defined by Sharma and Gupta (2018) and improved categorization of each app risk level using an ensemble ML technique. The risk factor defined by Sharma and Gupta (2018) are briefly described as follows:

High Risk Factor: In this category, the risk factor only applies to the permissions found in the sample of malicious apps and not in benign apps.

Medium Risk Factor: In this category, the risk factor is determined if the percentage of permissions in malicious apps is greater than or equal to the percentage of permissions in benign apps, this category has a medium risk factor.

Low Risk Factor: In this category, the risk factor is determined If the percentage of permissions in malicious apps in this category is lower than the percentage of permissions in benign apps in this category, then it is low risk.

No Risk Factor: In this category, the risk factor only applies to the permissions found in the sample of benign apps and not in malicious apps.

2.10 RESEARCH GAPS

Following the outcomes of the analysis of the literature presented in this chapter, it can be concluded that most of the proposed security solutions in the MCC environment are not sufficiently comprehensive as they only provide countermeasures to a few of the known security threats. In addition, and to the best of my knowledge, none of the existing security solutions proposed in the literature addresses all MCC's known security threats and requirements; the majority of the solutions only focus on user authentication and data storage, applying biometric and cryptographical approaches as a means of providing a secure MCC environment.

Furthermore, the analysis of the literature shows that activity monitoring using IDS techniques has the potential to address a wider spectrum of the threats that are faced by the MCC environment when compared to the more popular approaches such as user authentication and data storage

protection. While some of the current research in MCC security has investigated the application of ML methods in activity monitoring using IDS, the reported results still show a high rate of false alarms when detecting anomalous behaviour in MDs. In addition, the security algorithms employed by most existing ML techniques require significantly high computational resources, which makes their implementation infeasible at the MD level. The relatively high ML training time also affects the performance of these techniques.

Nevertheless, only a few studies have proposed a prevention system for managing intrusions in the MCC environment. However, these prevention techniques are passive (i.e., user attention is required). The need for an active prevention approach that requires an automatic response without user attention is still an open area for further research.

Overall, security issues affecting the MD node as part of the MCC environment, and the communication channels used by MDs have not received much attention in the extant research. However, attackers do focus their attention on the MD node as it is relatively more exposed (due in part to the lack of security awareness amongst MD users). Attackers target the MD node using malicious apps, code obfuscation, and repackaging of popular and legitimate apps with malicious payloads; these are difficult to detect by the defensive techniques currently available to MD users. As shown in the literature review, existing defensive techniques such as anti-virus and anti-malwares commonly used to protect the MD node in the MCC environments apply predominantly an SB detection approach, which cannot detect zero-day attacks on these devices. As already highlighted above, the security solution offering malware detection using ML techniques at the device level (i.e., without considering the other layers of the MCC architecture) may be computationally heavy and thus not feasible, given the resource constraints of the MDs. It was also observed that most of the existing solutions also require a constant connection between an online server and the MD consumption node for effective protection. However, it was shown earlier that in the MCC environment, the MD node is highly exposed, and, therefore, the security solution gaps identified above need to be addressed. Thus, the focus of this research is on the security needs and requirements of the MD node as part of the MCC environment. It aims to propose an implementable solution that offers better protection of the MD node in the MCC environment.

2.11 CHAPTER SUMMARY

In this chapter, detailed background studies are discussed in the CC, MC, and MCC environments. The service models for both CC and MCC technology were briefly discussed. The CC deployment model, together with the benefits and challenges of both the CC and MCC technologies, was presented. This chapter also discusses in detail the security issues facing MCC by highlighting the security requirements, vulnerabilities, and threats associated with the MCC technology. The analysis of the STRIDE model using the MCC threats category was also presented in this chapter. General background on intrusion detection in both mobile and cloud-based environments was also discussed in this chapter. The state-of-the-art in security and the current proposed solution that addresses security issues at the MD node of the MCC infrastructure were discussed in this chapter. In addition, data security frameworks and IDS proposed and developed for the CC, MC, and MCC environments were discussed and analysed. The results of the literature survey analysis were used to identify existing research gaps that determine the focus of this study. A solution that addresses the identified gaps are discussed in subsequent chapters.

CHAPTER THREE

DESIGN SCIENCE RESEARCH METHODOLOGY

The previous chapter presents a detailed review of the literature on data security frameworks and intrusion detection systems in the mobile and cloud environment. The detailed analysis of the literature highlighted that none of the existing frameworks addresses all the known security threats in the MCC environment. The literature also shows that the user layer is relatively more exposed to attacks than other MCC architecture layers. Therefore, improving the user layer's security in the MCC architecture is critical. This research addresses the user layer security issues by proposing a comprehensive framework that protect against the top security threats relevant to the MCC environment. The framework is used in the development of a proof-of-concept IDPS prototype system that significantly enhances the security of the user layer of the MCC environment.

This chapter presents the research methodology and research questions adopted for this study. The research question's solution was discussed in line with the research methodology, outlining the various steps required to complete each research phase. The proposed framework prototype system to enhance the security of the user layer of the MCC was also presented in this chapter.

3.1 RESEARCH METHODOLOGY

The Design Science Research Methodology (DSRM) was adopted as the research methodology required to conduct this study. This study aims to develop a prototype system as a proof of concept, as outlined in research objective three, in chapter one, section 1.3. Similarly, the study that focuses on the design and evaluation of an artifact, such as the proposed prototype system that is implemented as a proof of concept in this study, has used the following research methodology reported in the following works, for example, Hevner et al. (2004), Peffers et al. (2007), and Offermann et al. (2009). Therefore, the choice of this methodology in this study.

Peffers et al. (2007) stated that the DSRM is the framework that includes guidelines, practices, and procedures required to perform specific research that leads to the design of artifacts. The importance of this methodological approach is that it provides guidelines for the design and improvement of an artifact through continuous testing and iteration (Offermann et al., 2009). Hevner et al. (2004) introduced the DSRM in information systems (IS), which supports complex, artificial and purposefully designed systems. This methodology uniquely addresses the research problem by solving IS issues more efficiently. This research approach focuses on developing and evaluating artifacts for a specific research problem. The DSRM also highlights the researchers' contributions to the body of knowledge by analysing relevant literature and addressing issues raised in the gaps identified in this study.

In IS, to accomplish a better understanding of and application of DSRM as a research methodology, Hevner et al. (2004) suggest that design is both a “process” (i.e., a set of activities that are acted

upon by the world) and a “product” (i.e., an artifact that is sensed by the world). The DSRM supports a problem-solving concept that shifts perspective between design procedures and designed solutions continuously for solving complex problems in IS research. The process involved in the DSRM consists of a sequence of expert activities that produce an innovative design artifact. The evaluation stage of this method provides feedback information and a better understanding of a given research problem, which hence improves both the design process and the quality of the final product. There is a series of iterations between the distinct phases of this methodology before a final artifact is produced.

Hevner et al. (2004) mentioned four types of artifacts, i.e., constructs, models, methods, and instantiations. Most artifacts are designed to address unsolved problems, and their evaluation is based on their utility in solving those problems. Constructs provide the language used in defining the problem and solution. A model represents an instance of a real-world problem. Methods define the processes involved in solving a problem and provide guidelines on how the problems are solved. An instantiation shows the implementation of constructs, models, and methods in a working system. Based on the DSRM proposed by Hevner et al. (2004), instantiation is one of the most important artifacts to assess whether a solution or a prototype works since it adopts the implementation of constructs, models, and methods in solving the problem.

As the research involves building a software artifact., other potentially suitable software development methodology was also considered among them, prototyping, adaptive software development and model-driven engineering (Saeed, et al, 2019). However, these approaches are concerned to a significant degree with identifying user requirements and tailoring the artefact to meet these. The focus of this research was the development of a framework, with the software artefact representing an instance of the framework that demonstrated the feasibility of the framework's implementation. Adopting DSRM as thus study’s methodology allowed to both model the real-world threat environment as relevant to the context of the research, and to design "new means to...change and improve reality” (Venable et al, 2017).

This study applies the process mentioned in the DSRM by Offermann *et al.* (2009), as shown in Figure 3.1, to develop and implement a novel framework as a proof of concept to enhance data security in the user layer of the MCC architecture. The DSRM consists of rigorous steps required to design artifacts to solve a given problem, contribute to the body of knowledge, and provide a means to evaluate the proposed solution, as shown in Figure 3.1. The DSRM consists of three main phases (i.e., problem identification, solution design, and evaluation). Each of these phases is broken into different sequential steps that often refer to each other. The most crucial stage of the DSRM is designing an artifact that addresses the problems identified in the research.

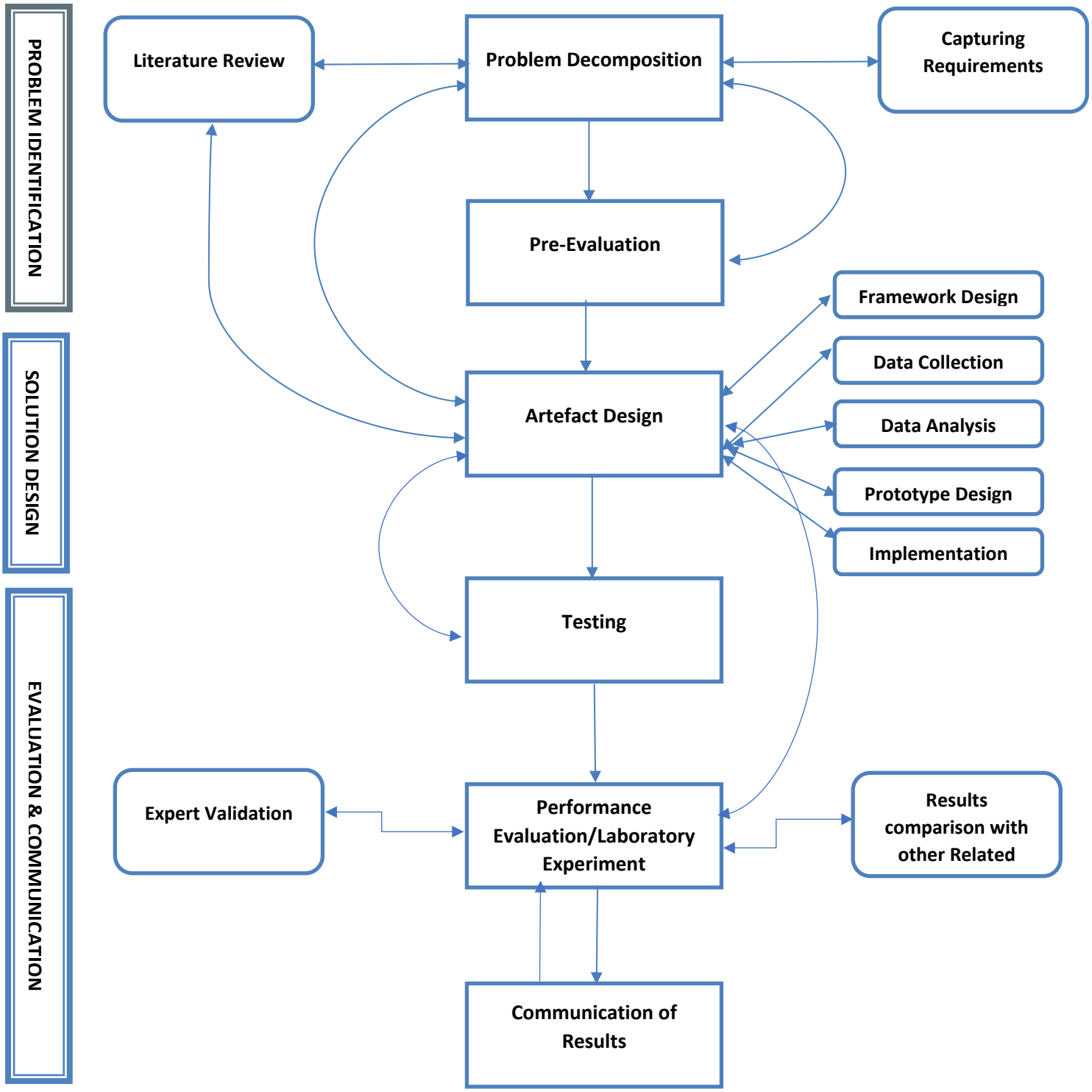

Figure 3.1: Research Process Followed in this Study

3.2 PROBLEM IDENTIFICATION PHASE

This study first reviews the existing data security framework to establish the MCC environment's state-of-the-art data security solution landscape. Second, a review of the IDS frameworks in the cloud-based environment to identify issues related to existing intrusion detection methods. The problems identified were then analysed to meet the research objectives. At the end of the problem identification phase, research gaps are identified, leading to the formulation of the research question.

Main Research Question: What security components are required in a framework that can protect MCC resources against attacks and enhance the security of user data in the MCC environment?

To address the main research question and meet the research goal stated in chapter one, section 1.3, this study provides answers to the main research question and formulates possible research sub-questions. Hence, the first research sub-question (RSQ) is formulated as:

> **RSQ1:** Which specific MCC resources require to be protected to enhance the security of this environment?

From the evidence shown in the literature analysis, existing research paid less attention to the user layer of the MCC infrastructure. The vulnerabilities associated with malicious apps can easily find their way into the devices used in the MCC environment because MCC Android users are allowed to download and install apps from official and unofficial app stores. The compromization of the user layer by malicious apps can serve as an entry point into the MCC environment, directly or indirectly affecting other MCC architecture layers. This study addresses these security issues by focusing on the mobile device as a resource that needs adequate protection in the MCC environment.

> **RSQ2:** What approach can be used to protect the identified MCC resources in RSQ1?

In the literature review, different approaches have been reported to protect MCC resources against attacks. The most dominant approach used in protecting MCC resources such as user data is the biometric and cryptographic approach. It was evident in the analyses of literature that these approaches are widely used in protecting MCC resources, although they only protect the MCC resources against few numbers of known security threats in this environment. A protection approach that offers more comprehensive protection to these resources is required. Therefore, this study adopted the use of the Intrusion Detection and Prevention System (IDPS) approach using ensemble ML technique. The IDPS when combines with ML approach offers protection to a wide spectrum of threat as reported in the literature review and can protect MCC resources against both known and unknown attacks.

> **RSQ3**: What metrics can be used to evaluate the performance of the identified approach in RSQ2 and how can this approach be implemented to protect the resource identified in RSQ1?

Based on the detailed analysis of the literature reported in chapter two, several metrics were reported, this study adopted the classification accuracy of an attack, precision rate, recall rate, false alarm rate and energy consumption as metrics that can be used for the evaluation of the prototype system.

3.3 SOLUTION DESIGN PHASE

The second phase of the DSRM focuses on the proposed solution design to address the problems identified in this study. The proposed framework and the prototype that will be implemented as a proof of concept are discussed as follows:

3.3.1 THE PROPOSED MCC DATA SECURITY FRAMEWORK

The solution designed in DSRM provides support for the design of artifacts that guide the design objectives. Evidence from the literature analysis suggests that an MCC data security framework that aims to offer a comprehensive security solution should address threats across the different layers of the MCC architecture. This study suggested that a comprehensive framework should address at least five or more known security threats that cut across the different layers of the MCC architecture. Therefore, this study recommends that these threat types T1, T4, T5, T7, T9, and T10 should be addressed at the MCUL layer, threat types T1, T2, and T8 at the MNCL layer, and threat types T1, T2, T3, T4, T5, T6, T7, T10, and T11 at the MCSPL layer.

This study proposes a comprehensive data security framework that protect MCC resources' confidentiality, integrity, and availability to their users, as shown in Figure 3.2. The proposed framework contains security management models that handle data security at rest and during transmissions across the three layers (MCUL, MNCL, and MCSPL) that make up the MCC architecture. The proposed framework adopts different existing security approaches to build the security management models across all three layers of the MCC infrastructure.

A. ***MCUL Security Model***: The security management model at the user layer applies both static and dynamic analysis of mobile devices by monitoring the behaviour of apps that reside on the device using an ensemble ML technique to detect malicious activities at the MCUL layer. The security management at the user layer uses an IDPS approach to handle the detection of malicious activities and the Attribute-Base Encryption (ABE) technique to protect the data stored at the user layer in the MCC environment.

B. ***MNCL Security Model***: The security management model at the MNCL of the proposed framework applies an ML approach to monitor the user network requests coming from the devices in the MCC environment and automatically stops all malicious requests detected from the MCC user devices. The intelligent model deployed at the MNCL

reduces the number of external attacks on the MCC resources located in the cloud infrastructure.

C. ***MCSPL Security Model***: The security management model that resides on the MCSPL applies two levels of protection to tackle attacks coming from an external source using an Identity Management System (IdMS) and an ensemble ML-based IDPS position on the cloud network. The IdMS handles all the user authentication and authorization requests to manage user access to MCC resources at the MCSPL. The ensemble ML-based IDPS stops all malicious activities that go undetected at the MNCL to protect MCC resources at the service provider layer. The proposed framework provides an ensemble ML-based IDPS deployed on each virtual machine and a hypervisor-based IDPS to protect the cloud infrastructure against insider attacks. The proposed IDPS position on the service provider layer is complemented with malware/spyware software to detect malicious cloud-based applications on the MCSPL. The proposed framework provides an ABE component to handle data encryption at rest as a last line of defence.

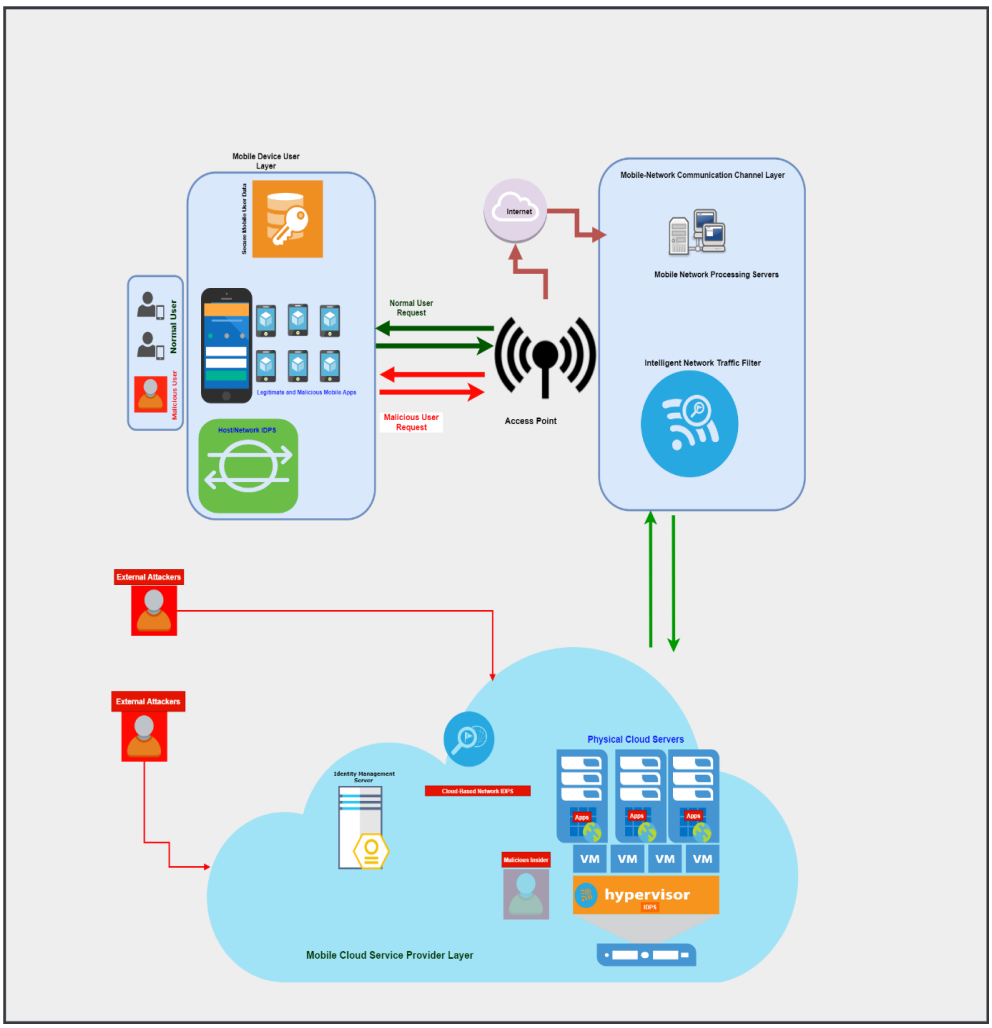

Figure 3.2: The Proposed MCC Data Security Framework.

3.3.2 THE PROPOSED PROTOTYPE SYSTEM

Based on the proposed framework in Figure 3.2, this study proposed implementing a prototype system as a proof of concept that offers a better security solution to the user layer of the MCC architecture. The proposed prototype system is a host-based Mobile-Cloud **In**trusion **D**etection and **Pre**vention **S**ystem named **MINDPRES,** shown in Figure 3.3**.** The proposed prototype system (MINDPRES) aims to analyze the behaviour of different apps that reside on the MD by monitoring the app activities both when the user is using the device and when the device is idle. MINDPRES has an application evaluator that evaluates an application's risk to determine its risk level. When MINDPRES is installed on a device for the first time, it analyses each user-installed app, computes their risk score value, and determines each app's risk category. MINDPRES placed the list of apps that fall into high and medium risk categories into the app watchlist to constantly monitor each app's network activities within and outside the devices. The installation of a new app on the MCC user device triggers MINDPRES to read the content of the manifest file of the newly installed app and offload the requested permissions and intent by the app to the cloud server for preliminary risk

assessment. The risk level of the app is evaluated by an ensemble ML model using the number of dangerous permissions, intents, and hardware components required by the app for its basic functionality. The cloud-based app evaluation uses a static approach to determine the app's risk level, assign a risk score value (i.e., high, medium, and low), and send a response to the device based on the app's assessment results. The prototype system prioritizes monitoring apps behaviour with high and medium risk levels at the device level.

Furthermore, MINDPRES uses a Host-Based-IDPS to safeguard the device by analysing suspicious API calls made by its apps. The information extracted by the Host-Based IDPS at runtime is used to monitor the device for malicious behaviour. The prototype system applies a hybrid detection method to monitor apps network activities (i.e., Internet usage, requested URL, upBytes, DownBytes, etc.). The prototype system monitors the device activities when the user is active (using the device) and when the user is not active (not using the device).

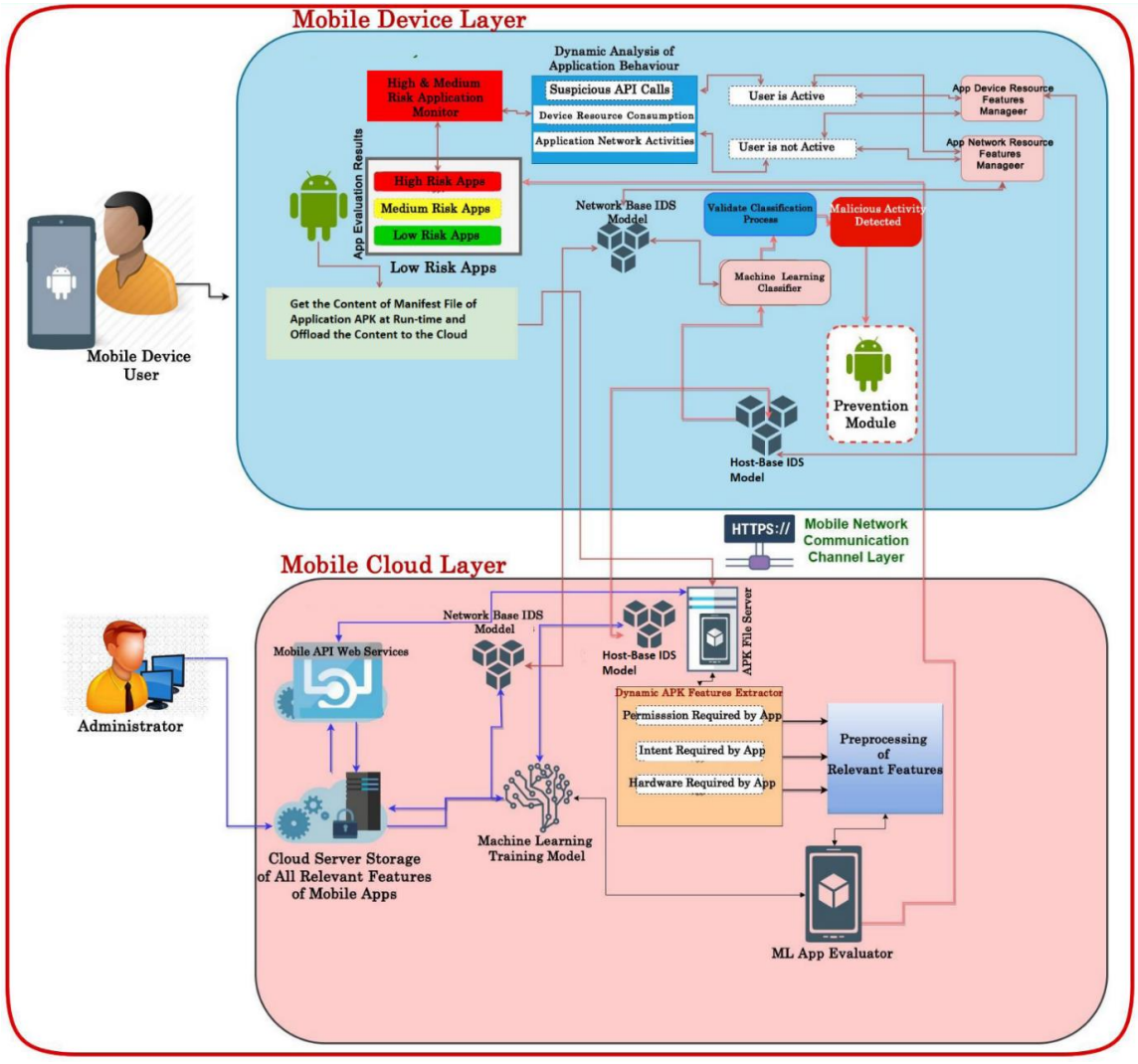

Figure 3.3 The Proposed Prototype System (MINDPRES)

The proposed prototype system also applies an efficient ML feature selection (FS) technique, which removes features that are not important for the classification of malicious apps for improved performance. The FS approach removes features that are not important in distinguishing between malicious and benign apps. The process also involves selecting features (such as the required permissions, required intent, and required hardware used) common to malicious and benign apps. The statistical analysis of these feature sets alongside the ML FS algorithm identifies the feature sets that are important for use in the training of the ML model.

Legitimate apps repackaged with malicious payloads by an attacker, when installed on the user devices are hard to detect by existing security solution in extant literature. The proposed prototype system monitors the apps' activities at runtime, both when used and when not used. The proposed prototype system includes an automatic intrusion prevention module to block detected malicious network traffic. However, MINDPRES gives users the flexibility to either execute such an app if the user feels the app is safe for her device due to the possibility of false alarms. In this study, the issues associated with the infeasibility of running an advanced security system using ML algorithms at the device level are mitigated by training the ML model at the cloud end and deploying the trained model to the device. Secondly, the application evaluator assesses each mobile app's risk level at the device end. This addresses the MD resource constraints, and these devices enjoy the benefits of ML. This study also applies the ML algorithm, requiring less computation resources for the actual detection at the device level, even though a highly secure algorithm has been used at the cloud end. This technique, adopted by this study, solves the problem of constant connection to the cloud server for actual device protection, as offered by the majority of the existing solutions in the CC and MC environments.

3.4 Evaluation and Communication Phase

The last stage of the DSRM involves the evaluation of the artifact, which is the prototype system. The evaluation phase applies standard security policy benchmarks to determine the system's effectiveness. The evaluation procedure requires laboratory experiments and evaluation of the prototype system using Android mobile apps on real-life devices. System testing includes continuous iterations to meet the requirements of the specified artifact as contained in the solution design stage.

The effectiveness of the proposed prototype system was evaluated with a testbed of over 1000 mobile applications that was collected from Google Play Store and Android malware repository. The evaluation of the prototype system involves real life experiment (five Android devices) and simulation experiment with Android device emulators by installing the prototype system (MINDPRES) in the various devices. The experiment was carried out for two weeks on using five real-life devices and twenty devices' emulators.

The applications installed in each of the devices were tested on VirusTotal engine to ascertain whether the app is benign or malicious. The power consumption was also recorded for each stage of this experiment. The power consumption recorded during the experiment represent the battery usage when MINDPRES was installed on a real-life device.

After completing the testing of the proposed system, the evaluation of the prototype system follows. The evaluation plans include experiments with both real-life devices and device emulators to check for the effectiveness and efficiency of the prototype system.

Finally, a performance evaluation is conducted using Android mobile apps on real-life devices. This study uses the classification accuracy of an attack, precision rate, recall rate, false alarm rate, and energy consumption metrics to evaluate the performance of the prototype system. The evaluation results are compared with other related works in the literature. The proposed prototype system is evaluated by invited industry experts working in IT security roles.

3.5 CHAPTER SUMMARY

This chapter discusses the DSRM methodology and process adopted for this research. The DSRM guidelines for the conduct of this research were also discussed and applied to each of the stages. The solution design stage of the DSRM in this study discusses the proposed framework and a detailed description of the prototype system. The chapter also briefly discusses the prototype design plan and how the system will be implemented and evaluated.

CHAPTER FOUR

DATA COLLECTION, ANALYSIS AND LABORATORY EXPERIMENTS

The previous chapter discusses the research methodology adopted for this study. The research questions formulated were also addressed to align with the proposed framework and prototype system. The design, implementation, and evaluation plan of the proposed prototype system that will be implemented as a proof of concept were described.

This chapter presents a brief description of Android OS's security system and a detailed description of the dataset used in this study. The data collection and analysis of the retrieved apps metadata (permissions and intents) from the dataset is presented in this chapter. The metadata analysis of the dataset shows the permissions and intent that are commonly used by both malicious and benign apps. The analysis of the permissions and intents also reveals some specific permissions and intent that can be used in ML algorithms as features to differentiate between the two app categories. This chapter also describes the feature selection techniques used in related studies and how these techniques can be used to reduce dataset dimensionality. Furthermore, this chapter proposes a filter-based feature section method that identifies the set of features needed for successful detection of malicious apps. The proposed filter was applied to the dataset constructed in this study and used to inform the dynamic app feature extraction process that provided the input to the ML detection model.

In addition, a detailed description of the classification algorithms used in the experimental work and the evaluation metrics used to evaluate the performance of the proposed machine learning models is also provided. This chapter also reports the experimental work carried out to extract the dynamic behaviours of apps. The extracted data were used in the experiment, proposing two different machine learning models that use both static and dynamic feature analysis approaches. This chapter also presents the analysis of the collected data and the three laboratory experiments conducted. The results of the three experiments are also presented and discussed.

4.1 ANDROID OPERATING SYSTEM SECURITY DESCRIPTION

The choice of Android Operating System (OS) as a target environment for the implementation of the prototype system in this study is based on the very large market share currently occupied by this OS in the mobile market (statcounter, 2022). The statcounter (2022) has reported that the Android OS occupies over 71% of the current mobile market of MD users. The vulnerabilities introduced into the MCUL of the MCC environment, when Android user download apps from outside the official mobile app store which turn out to be malicious, makes this mobile operating system (OS) platform a prime target considering the large market share of devices running in this environment. However, the core security component of the Android OS comprises different layers that enable developers to build secure applications. The Android OS provides a secure system such that no app can perform

activities that may affect other apps on the device, using the application sandbox and permission-based access control system. Android uses Linux's user-based protection to identify and isolate application resources. The sandbox isolates applications and protects both the applications and the system from malicious applications. In the Android mobile OS environment, each application is assigned a unique user ID (UUID) and runs in a separate sandbox. Since the application sandbox is embedded in the OS kernel, this security model applies to native code and operating system applications. The application sandbox contains all software above the kernel, such as OS libraries, application frameworks, application runtimes, etc. Specific platforms restrict developers to a particular development framework, set of APIs, or language. On Android, there are no restrictions on the way an application may be written that are necessary to enforce security; native code is just as sandboxed as interpreted code in this regard (Android, 2021).

The current permission-based access control model adopted by the Android OS plays a vital role in the Android security system and provides a secure environment. It handles access control to sensitive device resources such as user data and the device's functionalities. The permission-based access control model feature of the Android OS is used to invoke API calls for various functionalities in the mobile ecosystem. A complete list of all the different permissions required by the app is declared and stored in the manifest file before installation (Idrees et al., 2017). Any app that requests communication with another app or access to sensitive device resources must have adequate permission to perform such tasks. However, the effectiveness of this security model depends on the mobile user's ability to judge the permissions that need to be granted to a specific application. The granting of permissions by the user to apps whose developers' intention is maliciously unknown to the user can expose the device to an attack and possible leakage of sensitive information from such devices (Rai, 2013; Alshehri et al., 2019).

Android permissions are classified into four categories based on their degree of protection: normal, dangerous, signature, and signature or system. Android includes an access mechanism that checks applications' permissions and determines whether they are authorized to access protected resources. Normal permissions are granted to apps without the user's intervention because they are not deemed harmful. For example, the user does not get any notification to authorize INTERNET permission since it falls under normal permission. The Android OS handles the authorization without user intervention. The dangerous permission requires user approval due to the risk of privacy leaks and access to sensitive API calls. Dangerous permissions are higher-risk permissions (such as READ CALENDAR) that grant requesting applications access to private user data or device control, negatively impacting the user. Malware developers can easily exploit these sets of dangerous permissions to target the end-users of this mobile environment. Signature permissions are granted only to apps that are signed with the same certificate that defines the permission. Preinstalled apps

or those signed with the device manufacturer's certificate are granted signature or system permissions, and third-party apps cannot access these permissions (Idrees et al., 2017; Feng et al., 2019).

The Android permission-based access control system requires users to grant requested permissions in two different ways (install-time permission requests and runtime permission requests). The older versions of the Android OS (Android 5.1.1 and below) enforce install-time permission requests, while the newer versions (Android 6.0 and above) apply runtime permission requests. The install-time request requires the user to grant all requested dangerous permissions before installing the app on the device. The install-time permission request restricts access to system resources and private data. This version of the permission-based security system is more vulnerable to attacks since the user had no choice but to grant the app all requested dangerous permissions during installation; otherwise, the installation process would terminate abnormally. Similarly, some developers attempted to enlist the required permissions for users' consideration when using the apps. Still, users are not likely to judge better because they do not understand the inner workings and implications of the app. These increased the risk of possible attacks due to security risks incurred from breached apps (Feng et al., 2019).

The run-time permission request model of Android 6.0 and above requires the user to either grant or deny an app some specific dangerous permission when the request is made for the first time. One of the issues associated with this permission request is that an app can be granted overprivileged permissions because some single dangerous permission handles different sets of APIs. For example, an app might request access to READ SMS permissions. Granting such permission to the app will result in overprivileged access because the READ SMS permissions belong to the SMS-related group permission. The approval of an app to use SMS permissions will automatically allow the app to use other SMS related permissions (such as SEND SMS and RECEIVED SMS) without requiring the user to either grant or deny the requested permission. In addition, an app might request the READ PHONE STATE permission if such permission is granted to the app. The app can access different types of information, such as the IMEI, SIM card serial number, SIM card operator, and device ID. These overprivileged issues associated with the run-time permission requests allow an app to access more resources than anticipated (Alshehri et al., 2019).

Android uses Intents to enable applications to communicate with one another in a loosely coupled fashion. Android Intent is a security mechanism for inter-component communication within the OS. Intent handles app access control to the resources of other apps on a device. Intent works with Android's permission-based access control system, preventing an app from gaining direct access to other apps' data without having appropriate authorization. The Intent communicates the intention of

an app to perform a specific action. The intent filter defined in the app manifest file communicates the type of Intent an app can receive. Intent are of two types: explicit and implicit. The explicit Intent requires launching a specified component when such a request is processed. In contrast, implicit intents allow the system to look for an appropriate component by looking at the various intent filters (Idrees et al., 2017).

The Android intent handles all the inter and intra- app messages securely. Intra-app communication takes place between different activities within the app domain. For example, an app might consist of different pages; users move from one activity to another, i.e., from one page to another. Also, an activity might involve a different page element on a single page, such as a button, textbox, etc. The Android intent allows developers to perform communication or interactions amongst these activities. In a more precise term, the intent is used to push data between different activities and carry the results at the end of a specific activity to another activity (Feizollah et al., 2017). Inter-app communication requires sending messages or data to other apps with the same intent. In this regard, the app must have declared an appropriate intent and granted the necessary permission to share data between various apps. An app must define what type of intent it can accept in the intent filter section of the app manifest file. The binder makes communication between various apps possible in the Android security system. The binder is responsible for all inter-process communication within the OS (Feizollah et al., 2017).

4.2 DATA COLLECTION

This study requires data collected from the manifest files of Android Package Kit (APK) installation files of Android apps to build an ML model. The ML model was used to develop the proposed prototype system (MINDPRES) to tackle the data security issues caused by malicious apps on MCC user devices. To construct the required dataset needed to develop the ML models, benign and malicious apps were downloaded from AndroZoo and RmvDroid repositories (Allix *et al., 2016;* Wang et al., 2019).

AndroZoo is one of the research community's largest repositories of Android apps. It contains over three million apps that belong to different app stores. The AndroZoo repository includes benign and malware apps drawn from the following app stores: Google Play Store, App China Store, Anzhi, AnGeeks, 1mobile, torrents, Fdroid, HiApk, Genome, Proandroid, Apk_bang, and Slideme. The apps in the AndroZoo repository were labelled based on the results of over 50 different anti-virus engines as either malware or benign apps using the VirusTotal service. The VirusTotal service (https://www.virustotal.com) is an online tool that contains tens of anti-virus engines. This service is utilized to scan files to determine if a file contains malicious code or not. The APKs in the AndroZoo repositories were classified as malicious if any of the anti-virus engines (at least one anti-virus

engine) in the VirusTotal service used for scanning the APK detected any malicious codes or malware samples. This approach resulted in the labelling of over 22% of apps collected from the official Android market store “Google Play Store” as malicious. In addition, using the results obtained from at least ten anti-virus engine results as criteria to determine if an app should be classified as malicious shows that 1% of Google app stores have malicious content or malware. The official market store has its own inbuilt security to analyze apps before uploading them to the store for the public to use. However, malware developers have developed new techniques to bypass the security checks of the official Android app store, which resulted in 1% (Google Play Store) of the apps being reported as malicious in AndroZoo repository.

The apps in the RmvDroid repository contain over 9,000 malware samples that belong to 56 malware families. However, the malware families of the malicious APK samples obtained from the AndroZoo repositories were not reported. The APKs contained in the RmvDroid repository were collected based on the results of the maintenance report of the Google Play Store for four years. Each year, a snapshot of the Google Play Store is created; the metadata for each app includes the name, description, developer name, number of user installs, user rating, and the app API. RmvDroid reported having crawled over 1.5 million apps in four years. The malicious APKs were determined by the list of apps removed from the Google Play Store by comparing each snapshot. The removed apps' APKs were scanned using the VirusTotal services to ascertain how many of such apps' APKS have been flagged by various antivirus engines as malicious and used the AVClass to assign each app APK a malware family label (Sebastián et al., 2016).

The collection of app APKs from the two repositories requires the authors' authorization to download the APKs because the APKs are not publicly available. The authors provided the authorization key to download the apps' APK from their repositories (AndroZoo and RmvDroid) using a C# console app design in this study. The APK files were automatically downloaded and stored on the university's remote server allocated for this research for further processing. Due to the size of the APK files, the APK files collection stage consumes a lot of time and requires a lot of storage space. Between June 1st and July 31st, 2020, over 40,000 unaltered app APKs were downloaded from both repositories.

This research uses the VirusTotal services to ascertain the cleanliness of the collected APKs. The APK files were scanned using the VirusTotal service engine to categorize each file as a benign or malicious app. In this study, each app APK is labelled “benign” if none of the anti-virus engines in the VirusTotal service flagged the APK file as malicious. In contrast, an app is labelled “malicious” if at least 15 VirusTotal anti-virus engines flag it as malicious. However, most research work in the literature has based their work on using 1 to 10 anti-virus engines as a criterion to determine if an app is malicious. This work adopted at least 15 “malicious” outcomes from the different anti-virus engines to classify an app as malicious to build a more reliable dataset. The reason for this required

number of anti-virus engines in the VirusTotal service (at least 15 anti-virus engines flagging an app as malicious) is to remove uncertainty and build a more reliable ML model. Although some of these apps might not be malicious, they might contain codes that perform some activities that compromise data security at the device level.

In this study, most of the benign apps labelled based were originally collected from the official Android app store, “Google Play Store,” as reported in AndroZoo. Most of the malicious apps that were labelled for the construction of the dataset were collected initially from the App China store in AndroZoo repository. After completing the scanning and labelling of the collected APK files, a total of 28,306 apps, 9,879 benign and 18,427 malicious apps APK, were labelled, while the remaining apps APK were discarded. In particular, the sampled malicious apps used in this study contain apps that use the Joker malware; the Joker malware is known to be quite pervasive and has affected several apps available in the Google app store. The distribution of the app APK source market used in constructing the dataset in this study is shown in Table 4.1.

Table 4.1 Apps Apk Source Market Distribution.

App Type	App Market	Total
Benign	play.google.com	9879
Malicious	play.google.com	2121
	angeeks	53
	angeeks|appchina	1
	anzhi	2403
	anzhi|appchina	1
	anzhi|mi.com|appchina	7
	appchina	12453
	appchina|mi.com	1
	appchina|play.google.com|PlayDrone	2
	appchina|VirusShare	2
	mi.com	5
	play.google.com	671
	1mobile	35
	play.google.com|appchina|PlayDrone	1
	play.google.com|PlayDrone	39
	play.google.com|PlayDrone|mi.com	1
	play.google.com|PlayDrone|VirusShare	1
	PlayDrone	93
	PlayDrone|play.google.com	49
	PlayDrone|play.google.com|appchina	1
	PlayDrone|VirusShare|play.google.com	1
	praguard	46
	slideme	22
	VirusShare	629
Total		28,306

4.2.1 DATASET CONSTRUCTION

The manifest file contains essential information (such as the required permissions and intents) about the app functionality. To construct the feature datasets required for the laboratory experiments conducted in this study, the collected app APK files were first pre-processed to extract the permissions and intents used by each app. The sets of unique permissions and intents used by the apps are shown in Appendix A, Tables 1 and 2). The various steps used in the pre-processing of app APKs in this research are presented in Figure 4.1. The APK of each label app was decompiled using the APK Easy tool. The APK Easy tool is a software utility tool use for decomiplation of an APK file. The APK Easy tool applies the reverse software engineering techniques to decompile the APK file and extract its manifest file. The Apk Easy Tool is a lightweight application that provides an excellent user interface that allows easy decomiplation of each apk file. The manifest file of each

app was extracted using a customized console program after completing the decomiplation process of each app's APK.

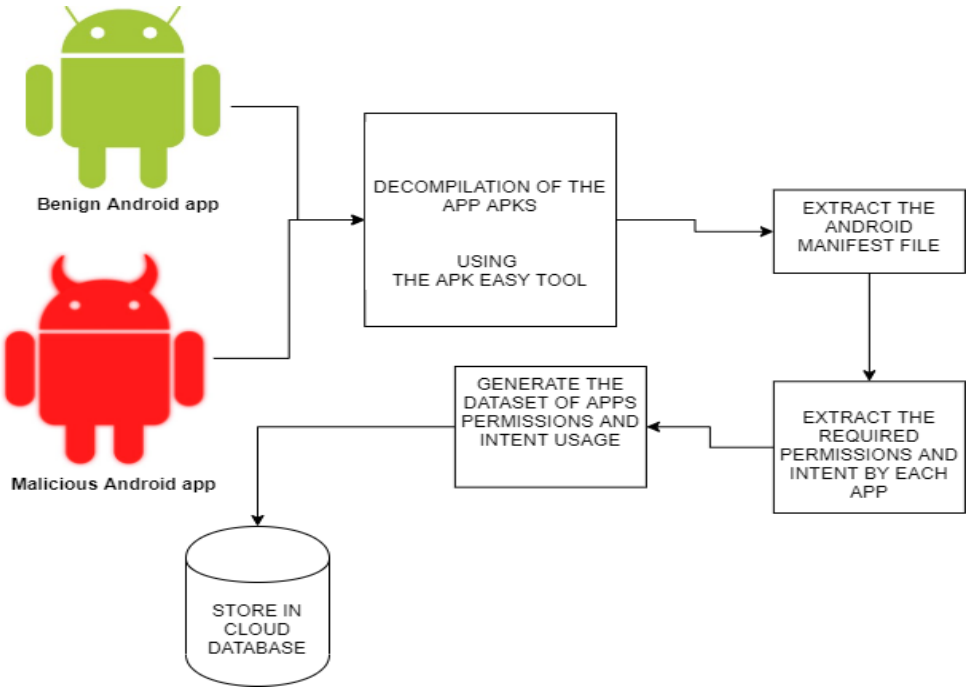

Figure 4.1 Data Pre-processing and Features Extraction Stages

The manifest file contains essential information (such as the required permissions and intents) about the app functionality. The three feature datasets used in the experiment (a feature set representing the permissions used by the apps, a feature set representing the intents used by the apps and combine feature set representing both the intents and permissions used by the apps) were constructed as explained below. In each dataset, the value of one represents that the app requires that specific permission or intent, and the value of zero indicates that the app does not require such permission or intent for its functionality. The final column in each dataset is the output column, which contains one or zero values. The value of one in the output column of the datasets indicates that the app with the corresponding permissions and/or intent features belongs to the malicious app category. The value of zero in the output column of the dataset indicates that the app belongs to the benign app category. The constructed dataset also contained the total number of permissions and/or intents required by each app and the hash value of the apk name, as shown in Table 4.2., Table 4.3 and Table 4.4.

The first dataset was constructed from the list of unique permissions in Table 1 in appendix A. using a binary vector. Such that $P_i = \{P_1, P_2, P_3, \ldots\ldots, P_n\}$ where n = 132 is the total number of unique permissions in the entire datasets.

Each app APK in the first dataset was represented using the binary vector of permissions required by an app as contained in the manifest file i.e., APP_i where

$$App(i)\begin{cases}1 \ if\ the\ permission\ is\ used\ by\ the\ app\ and\\ 0\ if\ the\ permission\ is\ not\ used\ by\ the\ app\end{cases}$$

The second dataset was constructed from the list of unique intents in Table 2 in appendix A. using a binary vector. Such that $I_i = \{I_1, I_2, I_3, \ldots\ldots, I_n\}$ where n = 131 is the total number of unique intents in the entire datasets.

Each app APK in the second dataset was represented using the binary vector of intents required by an app as contained in the manifest file i.e., APP_i where

$$App(i)\begin{cases}1 \ if\ the\ intent\ is\ used\ by\ the\ app\ and\\ 0\ if\ the\ intent\ is\ not\ used\ by\ the\ app\end{cases}$$

Finally, the third dataset was constructed by combining dataset 1 and dataset 2. The third dataset consists of unique permissions and intents in Table 1 and Table 2 in appendix A respectively using a binary vector. Such that $PI_i = \{PI_1, PI_2, PI_3, \ldots\ldots, PI_n\}$ where n = 236 is the total number of unique permissions and intents in the entire datasets.

Each app APK in the third dataset was represented using the binary vector of permissions or intent required by an app as contained in the manifest file i.e., APP_i where

$$App(i)\begin{cases}1 \ if\ the\ permission\ or\ intent\ \ is\ used\ by\ the\ app\ and\\ 0\ if\ the\ permission\ or\ intent\ is\ not\ used\ by\ the\ app\end{cases}$$

Table 4.2 Sample Structure of the Constructed Dataset 1(Permissions)

S/N	Apk	P_1	P_2	P_3	P_4	.	.	P_{132}	Total Permission Used	App Type
1	XXX	0	1	1	1			1	4	(1) Malware
2	YYY	0	0	1	1			0	2	(0) Benign
3	ZZZ	0	1	0	0			0	1	(0) Benign
4	TTT	1	0	1	0			1	3	(1) Malware
.										
.										
.										
28,306	WWW	0	1	0	1			0	1	(1) Malware

Table 4.3 Sample Structure of the Constructed Dataset 2(Intents)

S/N	Apk	I_1	I_2	I_3	I_4	.	.	I_{131}	Total Intent Used	App Type
1	XXX	0	1	1	1			1	4	(1) Malware
2	YYY	0	0	1	1			0	2	(0) Benign
3	ZZZ	0	1	0	0			0	1	(0) Benign
4	TTT	1	0	1	0			1	3	(1) Malware
.										
.										
.										
28,306	WWW	0	1	0	1			0	1	(1) Malware

Table 4.4 Sample Structure of the Constructed Dataset 3 (Permissions and Intents)

S/N	Apk	PI_1	PI_2	PI_3	PI_4	.	.	PI_{263}	Total Permission/Intent Used	App Type
1	XXX	0	1	1	1			1	4	(1) Malware
2	YYY	0	0	1	1			0	2	(0) Benign
3	ZZZ	0	1	0	0			0	1	(0) Benign
4	TTT	1	0	1	0			1	3	(1) Malware
.										
.										
.										
28,306	WWW	0	1	0	1			0	1	(1) Malware

Overall, the extracted metadata from the manifest files of all apps APK used to construct the datasets consists of 291,863 permissions and 112,022 intents. There were 132 unique permissions and 131 unique intents used in the dataset's construction. The number of apps APK that uses demanded each permission and intents as contained in the data retrieved from each manifest file are shown in Table 1 and Table 2 in appendix A.

4.2.2 PERMISSION AND INTENT USAGE ANALYSIS

This section discusses the analysis of the pre-processed data used to construct the dataset. The analysis of the dataset reveals how malicious apps utilize certain permissions and intents to steal sensitive information from the Android devices used in the MCC environment. First, the number of apps that use each permission and intent was computed. This computation shows the usage pattern amongst the two categories of app types (benign and malicious apps). As shown in Table 4.5, malware apps were seen to use more dangerous permissions than benign apps. The analysis of the dataset shows that out of the 132 unique permissions used by the entire set of apps, 110 of those permissions were common to both malicious and benign apps. The dataset analysis also reveals that most benign apps use fewer permissions than malicious apps. The dataset results show that benign apps use an average of 7 permissions, while malicious apps use an average of 13 permissions. However, some benign apps use more permissions like malicious apps. The results also reveal that the top ten benign apps use 36 to 64 permissions while malicious apps use 68 to 89. After analysing the results of the usage patterns of the permissions and intents demanded by apps in the dataset, the overall usage patterns of permissions and intents in the dataset analysis reveal frequently used permissions and intents amongst the collected apps in the dataset. Tables 4.5 and 4.6 show the top 25 permissions and intents commonly used by benign and malicious apps.

Table 4.5 Permissions Usage Analysis

ID	Permission Name	Category	Benign Usage (%)	Malicious Usage (%)
P1	WRITE_EXTERNAL_STORAGE	Dangerous	63.61	91.47
P2	READ_PHONE_STATE	Dangerous	25.84	96.52
P3	ACCESS_COARSE_LOCATION	Dangerous	24.95	68.20
P4	ACCESS_FINE_LOCATION	Dangerous	26.72	59.53
P5	GET_TASKS	Dangerous	6.49	50.17
P6	READ_EXTERNAL_STORAGE	Dangerous	30.58	33.42
P7	SYSTEM_ALERT_WINDOW	Dangerous	7.78	29.47
P8	READ_LOGS	Dangerous	1.85	30.57
P9	MOUNT_UNMOUNT_FILESYSTEMS	Dangerous	1.52	30.57
P10	CAMERA	Dangerous	19.34	19.70
P11	RECORD_AUDIO	Dangerous	8.31	20.18
P12	GET_ACCOUNTS	Dangerous	19.41	14.14
P13	CALL_PHONE	Dangerous	7.62	18.81
P14	WRITE_SETTINGS	Dangerous	5.70	16.50
P15	SEND_SMS	Dangerous	2.08	17.36
P16	INTERNET	Normal	98.80	99.89
P17	ACCESS_NETWORK_STATE	Normal	93.09	97.88
P18	ACCESS_WIFI_STATE	Normal	35.59	83.36
P19	WAKE_LOCK	Normal	58.29	45.16
P20	VIBRATE	Normal	34.04	50.92
P21	RECEIVE_BOOT_COMPLETED	Normal	23.30	38.22
P22	CHANGE_WIFI_STATE	Normal	4.92	31.42
P23	ACCESS_LOCATION_EXTRA_COMMANDS	Normal	1.00	22.65
P24	RESTART_PACKAGES	Normal	1.00	17.63
P25	MODIFY_AUDIO_SETTINGS	Normal	4.73	13.32
Note	Benign App Usage %= (Nos of Benign Apps that Use a Specific Permission /Total Benign Apps in the Entire Dataset) X 100 Malicious App Usage %= (Nos of Malicious Apps that Use a Specific Permission /Total Malicious Apps in the Entire Dataset) X 100 The total no of apps that uses a specific permission are listed in Table 1 in Appendix A For Example, P1(WRITE_EXTERNAL_STORAGE) The number of apps that uses P1 permissions in the dataset as shown in Table 1 in appendix A =6284 (**See S/N 128**) Therefore, Benign App Usage %= (6284/*9,879) X100=63.61* *Similarly for malicious app usage % for P1* (16855/18427) X 100 =91.47 ***Total Benign apps APK in the Dataset =9,879*** ***Total Malicious apps APK in the Dataset =18,427***			

Table 4.6 Intents Usage Analysis

ID	Intent Name	Benign App Usage (%)	Malicious App Usage (%)
I1	Action MAIN	99.86	98.59
I2	Category LAUNCHER	99.79	97.86
I3	Category DEFAULT	35.53	39.64
I4	Action BOOT COMPLETED	23.76	30.27
I5	Action PACKAGE ADDED	3.89	29.01
I6	Action VIEW	25.04	16.68
I7	Category BROWSABLE	22.72	14.19
I8	Action USER PRESENT	2.78	19.88
I9	Action PACKAGE REMOVED	1.98	14.15
I10	Category HOME	1.28	11.08
I11	Action SEARCH	4.66	2.24
I12	Action CREATE SHORTCUT	0.63	4.23
I13	Action MY PACKAGE REPLACED	6.58	0.16
I14	Action SEND	4.07	1.25
I15	Action PACKAGE REPLACED	2.08	2.25
I16	Action MEDIA MOUNTED	0.63	2.30
I17	Category LEANBACK LAUNCHER	3.83	0.36
I18	Action NEW OUTGOING CALL	0.69	2.02
I19	Action MEDIA BUTTON	3.21	0.44
I20	Action PACKAGE INSTALL	0.80	1.71
I21	Action SCREEN ON	0.26	1.44
I22	Category MONKEY	0.12	1.38
I23	Action TIMEZONE CHANGED	1.58	0.59
I24	Action SCREEN OFF	0.27	1.16
I25	Category INFO	0.87	0.81
Note	Benign App Usage %= (Nos of Benign Apps that Use a Specific Intent /Total Benign Apps in the Entire Dataset) X 100 Malicious App Usage %= (Nos of Malicious Apps that Use a Specific Intent /Total Malicious Apps in the Entire Dataset) X 100 The total no of apps that uses a specific Intent are listed in Table 2 in Appendix A For Example, I1(Action Main) The number of apps that uses I1 intents in the dataset as shown in Table 2 in appendix A =9865 (**See S/N 1**) Therefore, Benign App Usage %= (9865/*9,879) X100=99.86* *Similarly for malicious app usage % for I1* (18168/18427) X 100 =98.59 ***Total Benign apps APK in the Dataset =9,879*** ***Total Malicious apps APK in the Dataset =18,427***		

In order to visualize the usage patterns of both permissions and intents, Figures 4.2 and 4.3 present the data in Tables 4.5.and 4.6 (i.e., the usage of the top 25 unique permissions and unique intents by benign and malicious apps, respectively), sorted by value. The analysis of permissions and intent usage shows that most malicious apps utilize most of the permissions that fall under the dangerous category compared to benign apps. Suppose these permissions are granted to an untrusted app on a user device. In that case, the resultant effects can lead to the exposure of such devices to malicious activities unknown to the user. For example, if the GET_TASK permission is granted to an app, such an app can read the list of other apps on the user's device. As shown in Table 1 in Appendix A, over 50% of the malicious apps in the collected data requested the dangerous permissions s P1, P2, P3, P4 and P5. These permissions were also requested by a number of benign apps. However, only 63.61% of the benign apps requested P1 while 91.47% of malicious app requested the same permissions as shown in Table 4.5; a similar pattern can be observed for dangerous permissions P2, P3, P4 and P5.

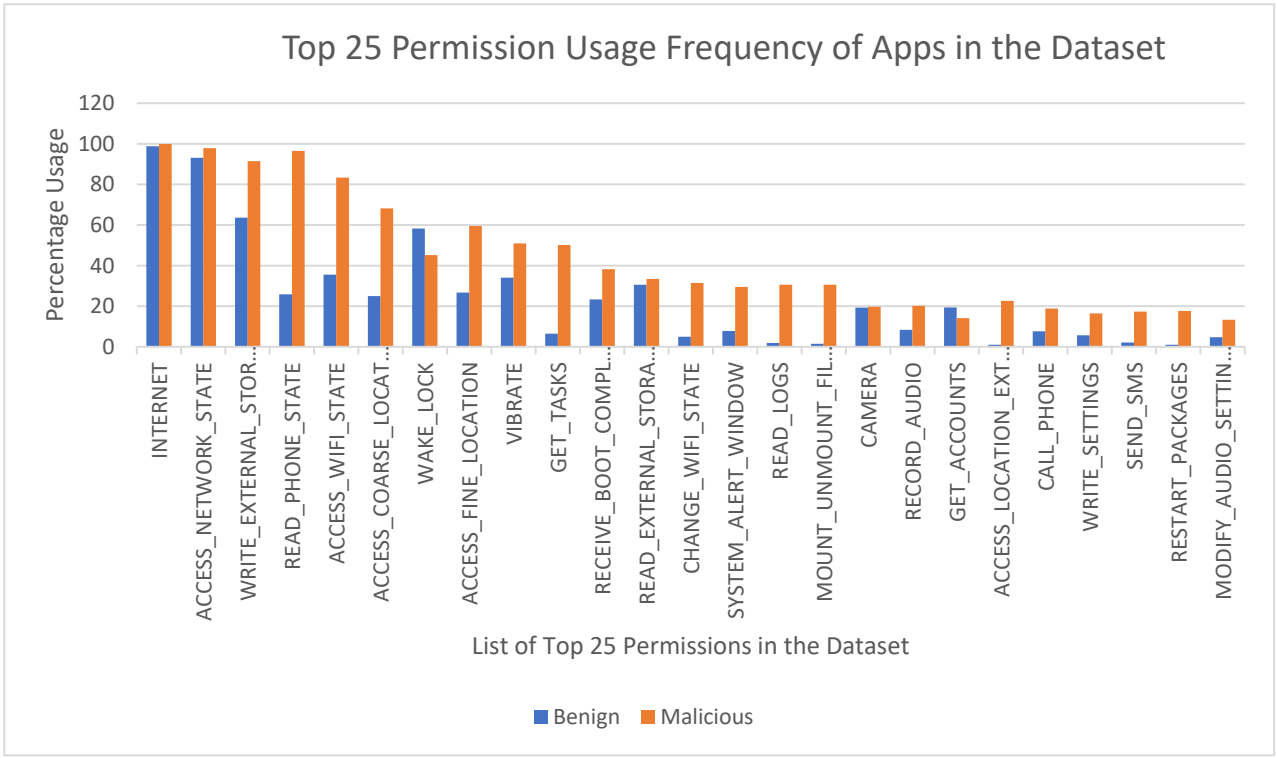

Figure 4.2 Top 25 Permission Usage Frequency of Apps in the Dataset

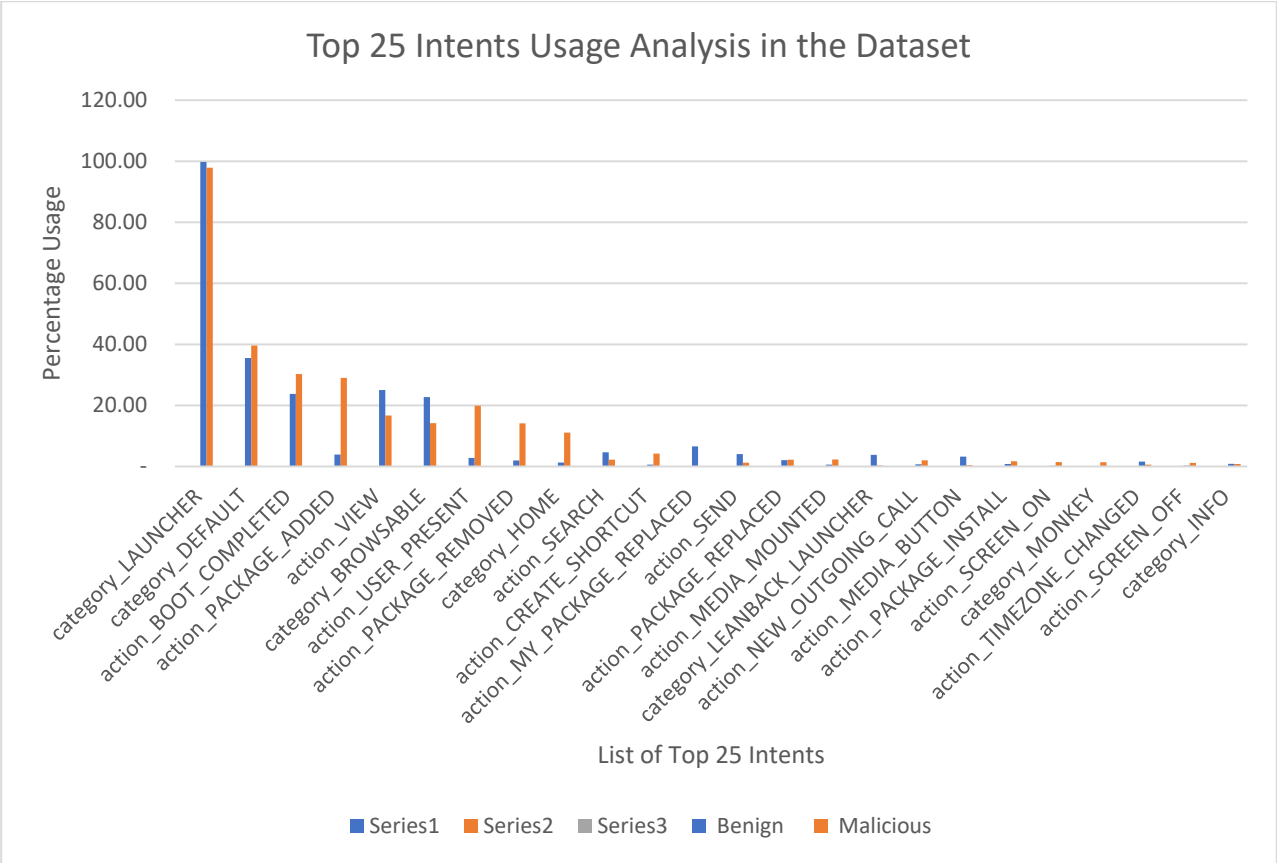

Figure 4.3 Top 25 Intent Usage Frequency of Apps in the Dataset

Further analysis of the results obtained in the dataset shows that 96.52% of the malicious apps in the dataset required the READ_PHONE_STATE permission. The READ_PHONE_STATE permission is the one most required by malicious apps. This permission enables hackers or malware developers to get the device identification number, such as the IMEI of a smartphone. Once granted to a malicious app, this permission results in targeted devices for specific online advertisements (ads). If not carefully reviewed by the user, these various ads mightily make their device more vulnerable. These vulnerabilities are exploited to target the devices and other devices connected to the same network as the infected device. The READ_PHONE_STATE permission, when used with other dangerous permissions, can enable apps to steal sensitive data or information from a device that is unknown to its owner. For example, an app requests both the GET_TASK and READ_PHONE_STATE permissions and receives approval from the device user. Such devices might be exposed to unauthorized access to sensitive information if such an app were installed from an untrusted source. However, some benign apps require both permissions, representing only a smaller fraction of the dataset. For example, 6.49% and 25.84% of benign apps only requested the GET_TASK and READ_PHONE_STATE permissions, respectively.

The analysis of most malicious apps' permissions and intent usage reveals the possibility of privacy leakage of user location unknown to the device user. Over 60% of malicious apps requested the ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION permissions. If these permissions are granted to an app, that app constantly monitors the user's locations. However, only 27% of benign apps requested such services. The analysis of the permissions and intents usage amongst the malicious APK shows that most malicious apps are fond of exploiting the READ LOGS permissions to monitor the activities of all apps on the user device. When the READ LOGS permission is used with the MOUNT_UNMOUNT FILESYSTEMS permission, malicious apps exploit the device's file system, leading to data security issues. The result in the dataset shows that 30.57% of malicious apps requested both READ_LOGS and MOUNT_UNMOUNT FILESYSTEMS permissions. Nevertheless, the issue of unauthorized access to sensitive information on devices is possible if an app request some of these dangerous permissions discussed while having access to the WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE permissions. If granted, these two permissions, together with other related dangerous permissions, enable an app to manipulate and access different storage media available on such devices and send that information remotely to the malware developers database.

In addition, 91.47% and 33.42% of malicious apps requested the WRITE EXTERNAL STORAGE and READ EXTERNAL STORAGE permissions, while only 63.61% and 30.58% of benign apps requested the same permissions, respectively. The possibility of unauthorized access to sensitive information stored on the devices is high since all Android apps access the INTERNET Permission. The results from the dataset used in this work show that almost 100% of malicious apps requested INTERNET permission, while 98.80% of benign apps requested INTERNET permission. In addition, most malicious apps tend to use a significantly higher number of dangerous permissions than normal permissions.

Furthermore, the issue of malware developers' targeting the intercommunication processing system of the Android OS using app intents to manage activities between app components and other apps residing on the device has raised serious security concerns. These techniques, used by some malicious apps, are hard to detect by the existing security system. The Intent feature of the Android OS enables both inter-and intra-process communications between different apps. Intent allows two apps to exchange information without user consent. Malware developers exploit this feature to steal sensitive information from the device. Based on the results presented in Figure 4.3. Intents as a means of communication between two apps unknown to the user were seen in some malicious apps in the dataset. Some malicious apps use the USER_PRESENT intent to determine when the device owner is using the phone and when the device is idle. This feature enables malicious apps to operate in the background since user activities may affect these processes. The Action_PACKAGE_ADDED

Intent allows an app to automatically add or load malicious code to existing apps or install new apps in the background without the user's consent. However, most such apps do have their icons disabled so that the user might not notice the presence of some illegal apps on their device. The BOOT_COMPLETED intent, when used with the RECEIVE_BOOT_COMPLETED permission, allows an app to monitor the boot status of the device. If an app with malicious intentions requests the RECEIVE_BOOT_COMPLETED permission, the permission is granted. Such an app can use the BOOT_COMPLETED intent to perform malicious activities that can affect the OS kernel of the device. Similarly, both benign and malicious apps used action_main and category_launcher intents to start their different activities on the device.

Finally, the Android OS allows users to grant or deny apps whenever they request permission from the dangerous permission set. The approval of some of these dangerous permissions to an app downloaded and installed from an untrusted source may expose the device to malicious activities. Because the use of intent with these dangerous permissions can easily compromise the device, the Android OS does not have the functionality to alert the user of any intent being used by an app. Most proposed systems have only considered permission as a static feature for designing malware detection systems. However, this study considers intent as a feature that can be combined with permissions to build an effective malware detection system. This research uses permissions and intents as static features to design a detection system to manage intrusion activities caused by malicious apps in the MCC environment.

4.3 LABORATORY EXPERIMENT 1

This experiment aims to find the best ML classification algorithms to distinguish malicious apps from benign apps. The best ML classification algorithms obtained from this experiment are used to build an ensemble ML model to manage intrusions caused by malicious apps in the user layer of the MCC architecture. This experiment uses the dataset constructed from the requested permissions and intents retrieved from the APK manifest files, as discussed in Section 4.2. The ML classification algorithms used for this experiment are shown in Table 4.7. The choice of these algorithms is based on the nature of the dataset constructed in this study. In addition, related works in this field of study has also adopted these set of ML classifier because of the nature of the proposed model to distinguish malicious apps from benign apps.

Table 4.7 The Selected ML Classification Algorithms

Classifier ID	ML Classification Algorithm
C1	Decision Tree Classifier
C2	Random Forest Classifier
C3	AdaBoosting Classifier
C4	Naïve Bayes Classifier
C5	Stochastic Dual Coordinate Ascent Classifier
C6	Multi-layer Perception Classifier
C7	K-Nearest Neighbours Classifier
C8	Linear Discriminant Analysis Classifier
C9	Logistic Regression Classifier
C10	Support Vector Machine Classifier

The ML experiment conducted in this study was implemented using the Python programming language using the ML classification algorithms (C1 to C10) in Table 4.7. The ML experiment uses the scikit-learn, pandas, and anaconda Python ML libraries. The experiment uses the three datasets constructed as explained in Section 4.2.1 (Tables 4.2, 4.3, and 4.4). Each of the three datasets was split in three portions (to be used for model training, validation, and testing, respectively), using the same proportion (Table 4.8). Each of the three datasets has the same distribution of apps APK in its dataset for training, validation, and testing, as shown in Table 4.8. The experiment was performed at the Auckland University of Technology, New Zealand, the WT Building laboratory using the following computer hardware configuration: Intel (R) Core (TM) i7-8700 CPU @3.20GHz, 16GB RAM, and a 500GB hard disk drive.

Table 4.8 Apps Distribution as used in the three datasets.

App Type	Training (72 %)	Validation (8%)	Testing (20%)	Total
Benign	7,096	788	1,995	**9,879**
Malicious	13,285	1,476	3,666	**18,427**
Total	**20,379**	**2,266**	**5,661**	**28,306**

4.3.1 MACHINE LEARNING CLASSIFICATION ALGORITHMS

The ML classification algorithms used in this experiment are well-known in related studies. Table 4 in the appendix A section shows a summary of related works and the ML classifiers used in their proposed solutions and experiments. The choice of these selected algorithms in Table 4.7 was based on the nature of the dataset constructed in this study and related works that have used similar algorithms in their various studies. The problem domain centres on classification problems. The model can differentiate between malicious and benign apps by learning from the features (requested permissions and intents by an app) represented in the dataset. These ML algorithms are briefly described as follows:

A. ***Decision Tree Classifier (C1):*** The Decision Tree (DT) Classifier belongs to the ML supervised family. This classifier is used for both classification and regression problems. This classifier's main goal is to create a model that can predict a class label for malicious or benign apps. The DT classifier uses a tree structure to solve a specific problem. The leaf node in the tree represents a class label, and attributes are defined in the internal node structure of the tree.

B. ***Random Forest Classifier (C2):*** The Random Forest (RF) belongs to the supervised ML family, and it is used for both classification and regression problems. The RF is an ensemble classifier that consists of multiple decision trees. The trees in the RF classifier learn independently on a subset of the training set that is randomly selected. The RF classifier uses bagging techniques to improve the overall results by combining the results of different learning models. The output of the RF ML classifier is determined by the most frequently occurring categories predicted by each learning model of each tree in the classifier. However, it is more suitable for multi-classification problems due to the nature of the tree, and it is also very effective when it comes to binary classification problems.

C. ***AdaBoosting Classifier (C3):*** The AdaBoost Classifier is an example of an ensemble ML classification algorithm belonging to the supervised ML family. This algorithm is used to create a strong classifier from a weaker one. The algorithms work on the principle that each learner is grown sequentially, such that each learner is grown from the previous learner except the first learner.

D. ***Naïve Bayes Classifier (C4):*** The Naïve Bayes (NB) classifier belongs to the supervised ML family, and the algorithms apply the probabilistic approach in solving classification problems using the Bayes' theorem. This ML model does not use iterative modelling like other supervised ML classifiers. When given a test sample, the NB classifiers compute the probability of various categories and determine the outcome by selecting the class with the highest probability.

E. ***Stochastic Dual Coordinate Ascent Classifier (C5):*** The Stochastic Dual Coordinate Ascent (SDCA) Classifier is an example of a supervised ML algorithm suitable for classification problems. This classifier selects random coordinates to maximize the dual objectives for solving a classification problem. The nature of its iteration is sequential as compared to other ML classifiers.

F. ***Multi-layer Perception Classifier (C6):*** Multi-layer Perception (MLP) belongs to the feedforward artificial neural network class. This classifier consists of three layers (input, hidden, and output). The classifiers provide a non-linear mapping of the input's vectors to a specific output vector. This classifier is applied to solving problems in different domains. The input layer is responsible for receiving the input data. The prediction of each class label is performed at the output layer. In contrast, the hidden layer acts as the computational engine

where all the necessary mappings occur before the output layer determines the results. The neurons in the MLP classifier are trained using backpropagation learning algorithms. This ML algorithm is very effective for solving classification problems.

G. K-Nearest Neighbours Classifier (C7): The K-Nearest Neighbours (KNN) classifier is a non-parametric classification algorithm that belongs to the supervised ML family. This classifier measures the distance between the test sample and the training samples and uses a majority voting concept to predict the category to which a specific sample belongs. Like the NB classifier, the KNN algorithms do not apply probabilistic concepts. The 'K' is the nearest neighbours that participate in the voting process. This type of classifier is more suited for classing new objects. The numbers of K can yield different results under the same circumstances.

***H.* Linear Discriminant Analysis Classifier (C8):** The Linear Discriminant Analysis (LDA) Classifier combines variables to maximize the difference between defined groups. This classifier is used when the predictors are distributed. This ML classifier solves both classification and regression problems. This classifier divides the dataset into k disjointed regions, representing the different class labels in the set. The final prediction is based on the maximum probabilistic allocation of the different class labels in the test set.

I. Logistic Regression Classifier (C9): The Logistic Regression (LR) Classifier is a supervised ML algorithm used for classification and regression problems. The LR classifier uses a statistical model that uses a logistic curve to fit the training dataset. Unlike other classifiers mentioned in this work, this classifier is easily updated to take new data. The classifier threshold can easily be adjusted. However, it requires a large sample size to be efficient in predicting. This classifier finds the probability that a given instance of an input set belongs to a specific class. The LR as a binary ML classifier requires a threshold to enable the model to differentiate between two classes.

J. Support Vector Machine Classifier (C10): The Support Vector Machine (SVM) classifier maps each input feature into an n-dimensional feature space such that n is the total number of features. This classifier identifies the hyperplane that separates each input feature into two different classes while maximizing the marginal distance for the classes and reducing the classification errors. This classifier can classify both non-linear and linear data items. This classifier uses the marginal distance between a class in its decision making.

4.3.2 VALIDATION METRICS USED IN THIS STUDY

This study uses the confusion matrix to validate the performance of the different ML classifiers in the experiment carried out in this study, which is defined as follows:

A. True Positive (TP): The total number of malicious apps that were classified correctly.

B. True Negative (TN): The total number of benign apps that were classified correctly.

C. False Negative (FN): The total number of malicious apps that were incorrectly classified as benign apps.

D. False Positive (FP): The total numbers of benign apps that were incorrectly classified as malicious apps.

4.3.3 EVALUATION METRICS USED IN THIS STUDY

To evaluate the performance of the different classifiers used in the experiment, the following metrics were adopted for this study based on the analysis of the 100 frameworks in the literature review as referenced in F1 to F35 in Table 2.5 and F36 to F100 in Table 2.6. Table 2.8 shows the definition of all the formulas used as evaluation metrics in this study which are obtained from the sources referenced in the reviewed works (F1 to F100). Furthermore, the relevant prior work used in the comparison of the results is referenced in Table 6.8, along with the evaluation metrics used.

A. Classification Accuracy (CA): This is the total percentage of the correctly classified malicious and benign apps in the dataset.

$$CA = \frac{TP+TN}{TP+FP+TN+FN} \times 100 \qquad \text{Eq. 4.1}$$

B. Error Rate (ER): This is the total percentage of all wrongly classified benign and malicious apps in the entire dataset.

$$ER = \frac{FP+FN}{TP+FP+TN+FN} \times 100 \qquad \text{Eq. 4.2}$$

C. Precision Rate (PR): This is the total percentage of correctly classified results of all malicious apps that belongs to the benign labelled in the dataset.

$$PR = \frac{TP}{TP+FP} \times 100 \qquad \text{Eq. 4.3}$$

D. Recall Rate (RC): This is the total percentage of malicious apps that are correctly predicted as malicious apps in the dataset.

$$RC = \frac{TP}{TP+FN} \times 100 \qquad \text{Eq. 4.4}$$

E. False Positive Rate (FPR): This is the total percentage ratio of malicious apps classified wrongly to the actual numbers of the malicious samples in the dataset.

$$FPR = \frac{FP}{FP+TN} \times 100 \quad \text{Eq. 4.5}$$

F. False Negative Rate (FNR): This is the total percentage ratio of benign apps classified wrongly to the actual number of samples in the dataset.

$$FNR = \frac{FN}{FN+TP} \times 100 \quad \text{Eq. 4.6}$$

G. False Alarm Rate (FAR): This is the total percentage average ratio of malicious and benign apps that are misclassified.

$$FAR = \frac{FPR+FNR}{2} \quad \text{Eq. 4.7}$$

H. F-Measure (FM): This is the harmonic mean of the proposed classifier which is obtainable from the value of both PR and RC.

$$FM = 2 \times \frac{PR \times RC}{PR+RC} \quad \text{Eq. 4.8}$$

4.3.4 Results Obtained from Experiment 1

The results obtained from laboratory experiment 1 conducted in this study were evaluated using the confusion matrix evaluation metrics discussed in Sections 4.3.3 and 4.3.4. The results obtained from the experiment conducted using the ten ML classifiers are presented in Table 4.9. The outcome of experiment 1 was also validated using a tenfold cross-validation technique, and the validation results are shown in Table 4.10.

Table 4.9 Results of the First Experiment

Classifier	Dataset	TP	FP	TN	FN	CA	ER	PR	RC	FM	FPR	FNR	FAR
C1	Permission	3451	201	1774	235	92.30	7.70	94.50	93.62	94.06	10.18	6.38	8.28
	Intent	3455	1238	723	235	73.93	26.07	73.62	93.63	82.43	63.13	6.37	34.75
	Both	3435	153	1842	231	93.22	6.78	95.74	93.70	94.71	7.67	6.30	6.99
C2	Permission	3520	186	1789	166	93.78	6.22	94.98	95.50	95.24	9.42	4.50	6.96
	Intent	3478	1248	714	212	74.16	25.84	73.59	94.25	82.65	63.64	5.75	34.69
	Both	3504	144	1851	162	94.59	5.41	96.05	95.58	95.82	7.22	4.42	5.82
C3	Permission	3489	296	1679	197	91.29	8.71	92.18	94.66	93.40	14.99	5.34	10.17
	Intent	3428	1244	717	262	73.35	26.65	73.37	92.90	81.99	63.44	71.0	35.27
	Both	3480	249	1746	186	92.32	7.68	93.32	94.93	94.12	12.48	5.07	8.78
C4	Permission	1043	61	1914	2643	52.23	47.77	94.47	28.30	43.55	3.09	71.70	37.40
	Intent	1042	86	1875	2648	51.62	48.38	92.38	28.24	43.25	4.39	71.76	38.07
	Both	1306	66	1929	2360	57.15	42.85	95.19	35.62	51.85	3.31	64.38	33.84
C5	Permission	3543	333	1642	143	91.59	8.41	91.41	96.12	93.71	16.86	3.88	10.37
	Intent	3414	1251	710	276	72.98	27.02	73.18	92.52	81.72	63.79	7.48	35.64
	Both	3488	248	1747	178	92.47	7.53	93.36	95.14	94.24	12.43	4.86	8.64
C6	Permission	3471	202	1773	215	92.63	7.37	94.50	94.17	94.33	10.23	5.83	8.03
	Intent	3463	1261	700	227	73.67	26.33	73.31	93.85	82.32	64.30	6.15	35.23
	Both	3461	164	1831	205	93.48	6.52	95.48	94.41	94.94	8.22	5.59	6.91
C7	Permission	3526	243	1732	160	92.88	7.12	93.55	95.66	94.59	12.30	4.34	8.32
	Intent	2439	610	1351	1251	67.07	32.93	79.99	66.10	72.38	31.11	33.90	32.50
	Both	3472	171	1824	194	93.55	6.45	95.31	94.71	95.01	8.57	5.29	6.93
C8	Permission	3552	403	1572	134	90.51	9.49	89.81	96.36	92.97	20.41	3.64	12.02
	Intent	3460	1289	672	230	73.12	26.88	72.86	93.77	82.00	65.73	6.23	35.98
	Both	3504	349	1646	162	90.97	9.03	90.94	95.58	93.20	17.49	4.42	10.96
C9	Permission	3490	268	1707	196	91.80	8.20	92.87	94.68	93.77	13.57	5.32	9.44
	Intent	3455	1276	685	235	73.26	26.74	73.03	93.63	82.06	65.07	6.37	35.72
	Both	3477	218	1777	189	92.81	7.19	94.10	94.84	94.47	10.93	5.16	8.04
C10	Permission	3512	291	1684	174	91.79	8.21	92.35	95.28	93.79	14.73	4.72	9.73
	Intent	3453	1273	688	237	73.28	26.72	73.06	93.58	82.06	64.92	6.42	35.67
	Both	3474	230	1765	192	92.55	7.45	93.79	94.76	94.27	11.53	5.24	8.38

Table 4.10 Ten-Fold Cross Validation Results of the Ten ML Classifiers

Dataset	Metrics	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Permission	Max CA	92.93	93.64	94.70	61.62	95.41	94.01	93.99	93.99	96.11	95.05
	MIN CA	87.99	91.17	89.75	50.53	85.51	89.05	88.84	86.57	89.40	88.69
Intent	Max CA	74.82	75.18	76.24	42.20	73.05	75.27	73.14	72.08	76.24	75.53
	MIN CA	69.50	69.86	69.15	37.46	57.60	70.57	65.37	67.08	69.96	69.96
Both	Max CA	93.64	95.41	94.70	53.00	94.35	95.76	94.70	93.64	95.41	95.41
	MIN CA	87.99	90.46	88.34	40.99	90.46	89.79	88.34	86.57	89.40	89.75

4.3.5 DISCUSSIONS OF RESULTS OBTAINED FROM EXPERIMENT 1

The results obtained from the experiment conducted using the ten ML classifiers are presented in Table 4.9. The results in Table 4.9 shows each ML classifier's performance. The results presented in Table 4.9 also show that combining both permission and intent in the ML model outperforms using either permissions or intents as a feature in the trained model. Although the results obtained using dataset 1(permissions as features) are similar to the results of the experiment using dataset 3 (permissions and intents as features). In addition, the results obtained using dataset 2 for the experiment (intent as features) were poor. However, the experimental results recorded with dataset 3 show that the ML model performs better when both permissions and intents are combined as features to build a malware detection engine, as shown in the results reported in Table 4.9.

Overall, the experimental results show that C2 as an ML classifier outperforms other ML classifiers as reported in the results, with a classification accuracy of 94.59%, a 96.05% precision rate, a 95.58% recall rate, and a 5.82% false alarm rate, as presented in Figure 4.4 and Figure 4.5. In addition, the C1 and C7 ML classifiers performed better in the experiments, as shown in Figure 4.4 and Figure 4.5. Both classifiers recorded a classification accuracy of 93.22 and 93.55, respectively. Therefore, the best three ML classifiers recorded in the experiments are C2, C7, and C1. The results of the ML experiment were validated using the ten-fold cross-validation ML techniques for each classifier. The ten-fold cross-validation results are presented in Table 4.10. The results show that the best three classifiers recorded in the experiment perform better as expected when validated using the ten-fold cross-validation technique presented in Table 4.10, even though classifier C9 using the permission dataset obtained the highest classification accuracy of 96.11%.

The best three classifiers (C2, C1 and C7) obtained 93.64%, 92.93% and 93.99%, respectively. The minimum classification accuracy for the best three classifiers obtained during the ten-fold cross-validation process is 91.17%, 87.99%, and 91.17%. The validation results obtained using dataset 3 (permissions and intents as features) achieve 95.41% accuracy for C2 and C9. The results show that the best three classifiers still perform better using dataset 3 than the results recorded in dataset 1 during the cross-validation process.

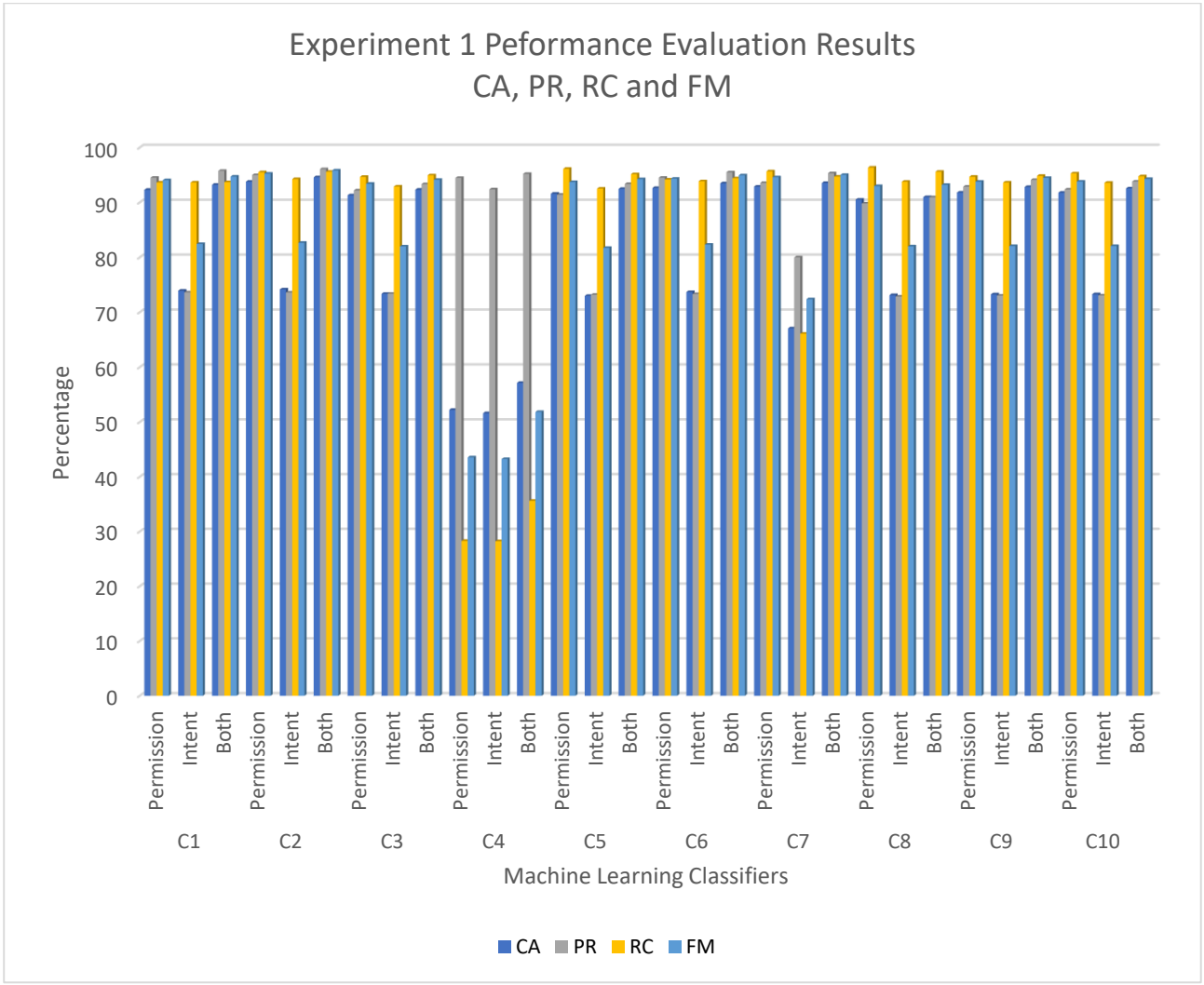

Figure 4.4 Performance Evaluation Results for the Ten ML Classifiers (CA, PR, RC & FM)

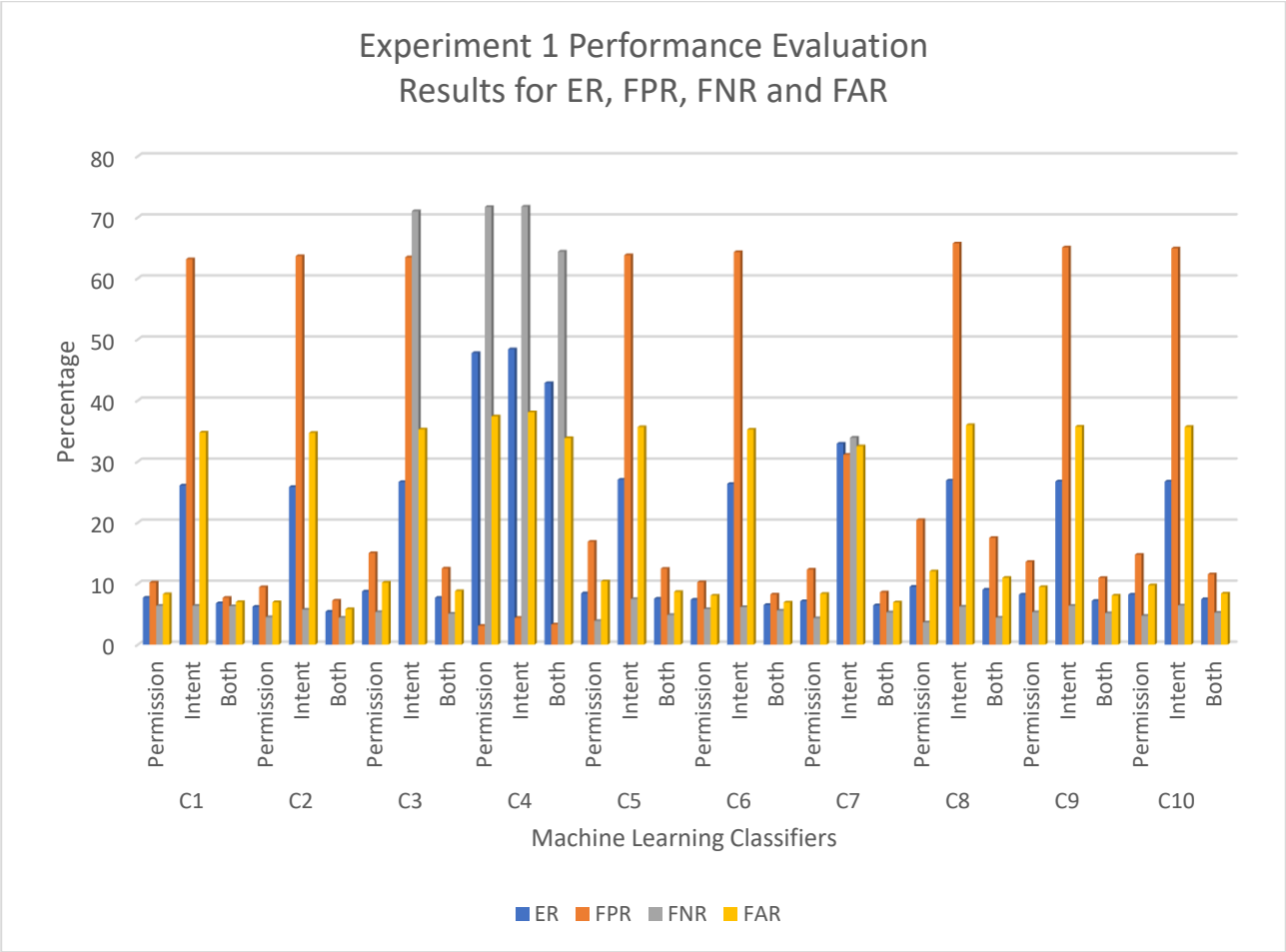

Figure 4.5 Performance Evaluation Results for the Ten ML Classifiers (ER,FPR,FNR, & FAR)

4.4 LABORATORY EXPERIMENT 2

This experiment aims to build an ensemble ML model using the best three ML classifiers recorded in the results obtained in experiment 1. The experiment uses dataset 3 discussed in Section 4.2.1. The experiment reported in this section was performed at the Auckland University of Technology, New Zealand WT Building laboratory using the following computer hardware configuration: Intel (R) Core (TM) i7-8700 CPU @3.20GHz, 16GB RAM, and a 500GB hard disk drive.

To improve the performance of the ensemble ML model, this study proposed a filter-based feature selection (FS) technique that uses a statistical approach to reduce the number of features needed to train the ensemble ML model. The proposed FS technique uses the analysis of the permissions and intent usage in the dataset, discussed in Section 4.2.2. as a basic criterion for the selection of features. The results of the ML experiment reported in this section are compared to the individual ML classifiers used in experiment 1 using the reduced feature set dataset generated after the FS process.

4.4.1 Feature Selection Methods

In this study, FS refers to selecting the features (from the entire features dataset) used in the ML model for malicious app detection. Selecting the most relevant features is critical because it improves the ML detection model's performance. The techniques for selecting the best features have been classified as filter, wrapper, and embedded feature selection methods (Şahin et al., 2021). While filter FS methods employ statistical techniques to select the best features, wrapper methods rely on heuristics (e.g., a machine learning classifying algorithm) rather than statistical techniques. While wrapper methods effectively determine the optimal feature subset, they are more computationally intensive than filter methods. Unlike filter and wrapper FS methods, which select the best features before training the ML model, embedded FS methods do so during the training process (Shabtai et al, 2012; Şahin et al., 2021).

In the existing literature, various feature types and FS techniques have been proposed. Most static analysis results published in the literature have been obtained using a small number of application samples, with only a few utilizing an ensembling technique to construct the malware detection model. For example, Shabtai et al. (2012) used Chi-Square, Information Gain (IG), and Fisher's Score as filtering methods; the set of features included CPU power consumption and Wi-Fi output volume. The three FS methods generated subsets of the 10 most significant features, the 20 most significant features, and the 50 most significant features, respectively. The authors' generic malware detection framework was evaluated using six different ML classifiers; the accuracy of malicious app classification ranged between 87 and 93%, with the Decision Tree (DT) algorithm achieving the highest rate.

On the other hand, the test set contained only a few (purpose-built) malicious apps. Alazab et al. (2020) used API calls as a feature set and achieved a classification accuracy of 98.10% by selecting relevant features using Chi-Squared. The proposed model reported in their work was evaluated using the real-world live dataset of 9,000 apps. This study proposed a filter-based FS method that selects features by applying a statistical approach, taking into account the relative usage of each permission or intent by benign and malicious apps, as discussed in the subsequent section.

4.4.2 The Proposed Filter-Based Feature Selection Technique

This section presents the proposed filter-based statistical use to select relevant features (using the full features in dataset 3). The high dimensionality (263 features) of dataset 3 used in experiment 1 can negatively affect the performance and energy consumption if such a feature dataset is used to train and build an ML model for resource-constrained MDs used in the MCC environment. This study proposed a new FS technique to help select relevant features (permissions and intents) to improve the performance of the ensemble ML model and make it feasible to be implemented in resource-constrained MDs in the MCC environment.

The proposed filter-based FS techniques use a statistical model to best perform features to build the ML model for effective malware detection. The statistical FS technique presented in this study uses a statistical Intrinsic Deviation Model (IDM) by considering the relative usage of each permission or intent by benign and malicious apps. This approach analyses the dataset and identifies usage patterns common to malicious and benign apps while identifying features used to distinguish between the two app categories. This statistical filter-based FS technique helps to eliminate some permissions and intent that are frequently used by both benign and malicious apps, and other features that are not relevant to the detection process are removed. For example, the INTERNET permission indicates that an app may access the internet services. The analysis discussed in Section 4.2.2 shows that almost all apps use INTERNET permission, irrespective of the app type. This kind of permission can affect the learning process of the model. Therefore, the need to remove such features and select the best features that the model uses to effectively differentiate between the two app types. The metrics formulated in equations 4.9 to 4.20 are used to define the evaluator (equations 4.21 and 4.22) that determine which features should be removed and which one should not be removed from the dataset; the evaluator aim to remove the features that are not relevant to the proposed detection model.

Let the set $PI = \{P_1, P_2, P_3, \dots, P_{132}, I_1, I_2, I_3, \dots, I_{131},\}$ be the non-empty set consisting of all the 132 unique permissions in the dataset , P_i, $1 \leq i \leq 132$, together with 131 unique intents in the dataset I_j, $1 \leq j \leq 131$. As such, the number of elements in the set $PI \equiv 263$.

Set

$$\phi_i = \frac{Total\ No.\ of\ apps\ uisng\ P_i}{Overall\ total\ No.\ of\ apps} \times 100\% \qquad \text{Eq. 4.9}$$

ϕ_i to be the percentage usage of each permission $P_i \in PI$ by (both benign and malicious) applications per overall total number applications in the entire data set, i.e., Eq 4.9

In a similar manner, the intent usage

$$\phi'_j = \frac{Total\ No.\ of\ apps\ uisng\ I_j}{Overall\ total\ No.of\ apps} \times 100\% \qquad \text{Eq. 4.10}$$

ϕ'_j to be the percentage usage of each intent $I_j \in PI$ by (both benign and malicious) applications per overall total number applications in the entire data set, i.e., Eq 4.10

Then, let a subset $T \subseteq PI$ called the testing set,

$$T = \{P_i, I_j \in PI: 3 < \phi_i \leq 90, 3 < \emptyset'_j \leq 90\}$$ Eq. 4.11

T containing some selected permissions and intents $P_i, I_j \in PI$ to be tested, such that P_i or I_j will belong to the set T if and only if the percentage usage of ϕ_i or $\emptyset'_j$ respectively, is strictly greater than 3% and less than or equal to 90%, i.e. Eq. 4.11

Now, corresponding to each permission, $P_i \in T$, define ξ_i to be the percentage usage of the permission P_i by benign apps per overall benign apps in the data set. That is,

$$\xi_i = \frac{No.\ of\ benign\ apps\ using\ P_i}{Overall\ No.of\ benign\ apps} \times 100\%$$ Eq. 4.12

In the same vein,

$$\xi'_j = \frac{No.\ of\ benign\ apps\ using\ intent\ I_j}{Overall\ No.\ of\ benign\ apps} \times 100\%$$ Eq. 4.13

Also, define η_i to be the percentage usage of the permission $P_i \in T$ by malicious applications per overall malicious applications in the data set. That is,

$$\eta_i = \frac{No.\ of\ malicious\ apps\ using\ P_i}{Overall\ No.\ of\ malicious\ apps} \times 100\%,$$ Eq. 4.14

and take

$$\eta'_j = \frac{No.\ of\ malicious\ apps\ using\ intent\ I_j}{Overall\ No.\ of\ malicious\ apps} \times 100\%$$ Eq. 4.15

Then let

$$\varepsilon_i = min\frac{(\xi_i, \eta_i)}{max(\xi_i, \eta_i)}$$ Eq. 4.16

be the ratio of the minimum to the maximum, respectively, of the two percentages ξ_i and η_i; and as well

$$\varepsilon'_i = min\frac{(\xi'_i,\eta'_i)}{max(\xi'_i,\eta'_i)} \quad \text{Eq. 4.17}$$

Furthermore, the metric function is simply defined by $\delta_i(\xi_i,\eta_i)$ measuring the difference between the percentages $\xi_i$and η_i for each permission $P_i \in T$, such that

$$\delta_i(\xi_i,\eta_i) = |\xi_i - \eta_i| = \begin{cases} \xi_i - \eta_i, \xi_i \geq \eta_i \\ \eta_i - \xi_i, \wedge\, \xi_i < \eta_i \end{cases} \quad \text{Eq. 4.18}$$

Similarly,

$$\delta'_j(\xi'_j,\eta'_j) = \left|\xi'_j - \eta'_j\right|. \quad \text{Eq. 4.19}$$

Conclusively, the evaluator $E(P_i)$ of each permission $P_i \in T$ based on whether or not the permission P_i is fit to be used as a relevant feature is given by

$$E(P_i) = \begin{cases} GOOD, if\ \frac{\delta_i}{min(\xi_i,\eta_i)} > \frac{\varepsilon_i}{2} \\ VOID, if\ \frac{\delta_i}{min(\xi_i,\eta_i)} \leq \frac{\varepsilon_i}{2} \end{cases} \quad \text{Eq. 4.20}$$

where “GOOD” implies that the permission $P_i \in T$ in question is indeed a relevant feature, while “VOID” therefore implies that the permission $P_i \in T$ in question is not such a good feature.

Similarly, the analogue of the evaluator $E(P_i)$ of each permission $P_i \in T$ being the evaluator $E(I_j)$ for each intent $I_j \in T$ as

$$E(I_j) = \begin{cases} GOOD, if\ \frac{\delta'_j}{min(\xi'_j,\eta'_j)} > \frac{\varepsilon'_j}{2} \\ VOID, if\ \frac{\delta'_j}{min(\xi'_j,\eta'_j)} \leq \frac{\varepsilon'_j}{2} \end{cases}. \quad \text{Eq. 4.21}$$

The proposed filter-based FS technique results in this study are presented in Table 3 in appendix A. The results of the FS technique show that out of the entire 263 feature sets, the proposed FS technique selected 39 feature sets comprising both permission and intent, whose status in Table 3 appendix A is indicated as "Good". The permissions and intent commonly used by both app types were identified as not relevant features as they contribute little to the detection process. The model was able to identify both high-risk permission and low-risk permission and intent for each app type.

Using only high-risk permission will result in the ML model being biased. The deviation approach involves features that contribute more to the detection and features that balance the selection process less. For example, the action_main intent and action_category launcher intent was used by almost 99% of each type. These kinds of features can mislead the learning model. The selected 39 features are presented in Table 4.11.

Table 4.11 The Selected 39 Features from the Proposed Filter-Based FS method

	Selected Features
Permission	PI3, PI4, PI5, PI9, PI31, PI33, PI38, PI39, PI41, PI47, PI52, PI55, PI62, PI68, PI71, PI79 , PI80, PI82, PI84, PI85, PI89, PI91, PI93, PI100, PI101, PI114, PI122, PI123, PI12, PI131
Intent	PI143, PI194, PI201, PI214, PI230, PI232, PI245, PI248, PI252

The selected features as presented in Table 4.11 were used in the design of the prototype system; these are labelled GOOD in Table 3 in Appendix A. Missing form Table 4.11 are the features that was removed based on the result of the proposed feature selection in the study; these 'missing features 'are shown in Table 3 Appendix A with the label VOID.

4.4.3 THE PROPOSED ENSEMBLE ML MODEL USING STATIC ANALYSIS APPROACH

This study uses apps' permission and intent requests as a static feature to build an ensemble ML model for detecting malicious user-installed apps in the MCC environment, as shown in Figure 4.6. The proposed ensemble ML model uses a supervised ML learning approach to train the detection engine. The detection engine uses voting ensemble classifiers. The voting ensemble ML classifier in the proposed model uses the best three ML classifiers, C1, C2 and C7 (outcome of experiment 1). The proposed ensemble ML model was trained using the 39 selected features obtained from the proposed filter-based FS method. The proposed ensemble ML model is used to detect malicious activities of users who install apps on their mobile devices at the user layer of the MCC infrastructure. The ensemble ML model uses static analysis of the permission and intent required for the app to perform its basic functionality to determine if the app is either malicious or not.

The ensemble ML model has an app permission filter used to extract permissions demanded by an app and generate the permission dataset. The permission dataset is made up of zeros and ones. The zero indicates that the app did not require specific permission, while the one indicates that the app demanded specific permission. Similarly, the model has an intent filter that reads the intent requested by an app, and the corresponding intent dataset is constructed. The ensemble model uses the union of the selected permission and intent based on the outcome of the selected feature in Table 4.11. as input for the model training. The classification outcomes of the proposed ensemble ML model depend on the majority votes among the three classifiers (C1, C2 and C7).

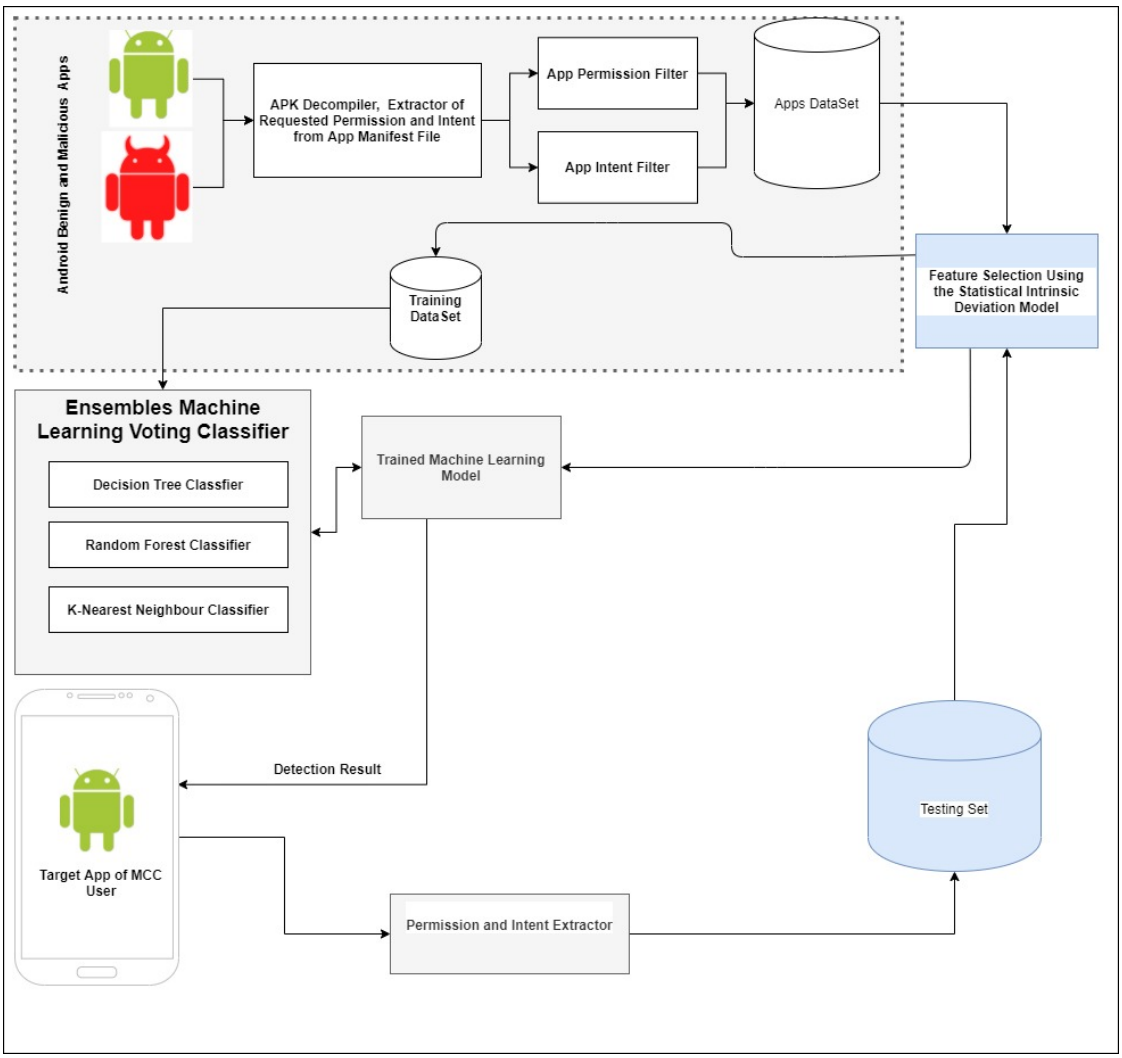

Figure 4.6 The Proposed Ensemble ML Model using Static Analysis Approach

4.4.4 RESULTS OBTAINED FROM EXPERIMENT 2

The results obtained from laboratory experiment 2 conducted in this study were evaluated using the confusion matrix evaluation metrics discussed in Sections 4.3.3 and 4.3.4. The second experiment uses the selected features outcome of the proposed filter-based FS technique to extract only the selected features columns from the original dataset 3 to form a new reduced dataset 4. Dataset 4 contains 39 features, with 1 output column. The second experiment uses the new dataset 4 to conduct an ML experiment. The experiment uses 11 ML classifiers (the proposed ensemble voting classifier and the original ten ML classifiers used in the previous experiments), and the results obtained are presented in Table 4.12.

Table 4.12 Results of the Second Experiment.

Classifier	TP	FP	TN	FN	CA	ER	PR	RC	FM	FPR	FNR	FAR
C1	3421	185	1816	240	92.78	7.22	95.29	93.44	94.36	8.45	6.56	7.50
C2	3496	174	1827	165	94.00	6.00	95.58	95.11	95.35	8.05	4.89	6.47
C3	3478	273	1728	183	92.39	7.61	93.14	95.25	94.18	12.84	4.75	8.80
C4	2699	224	1777	962	82.30	17.70	92.87	78.67	85.18	11.04	21.33	16.19
C5	3497	284	1717	164	92.53	7.47	93.24	95.36	94.29	12.64	4.64	8.64
C6	3470	205	1796	191	92.65	7.35	94.39	94.24	94.31	10.24	5.76	8.00
C7	3484	201	1800	177	93.50	6.50	94.80	95.17	94.98	9.55	4.83	7.19
C8	3524	412	1589	137	90.76	9.24	90.21	96.15	93.08	19.09	3.85	11.47
C9	3475	245	1756	186	92.88	7.12	93.93	95.14	94.53	11.24	4.86	8.05
C10	3483	270	1731	178	92.60	7.40	93.37	95.33	94.34	12.39	4.67	8.53
Proposed	3598	43	1958	63	98.13	1.87	98.82	98.28	98.55	2.15	1.72	1.93

4.4.5 DISCUSSIONS OF RESULTS OBTAINED FROM EXPERIMENT 2

The results obtained from the second experiment show that the proposed ensemble model performs better than the individual ML classifiers. The ensemble model results when tested with 2,001 benign samples and 3,661 malicious samples. The proposed model achieves the highest classification accuracy of 98.13%, with an error rate of 1.87%. The precision and recall rates were very stable as the model obtained 98.82% precision and 98.28% recall, as presented in Figure 4.7. The false alarm was less than 2%, showing that the proposed model can efficiently detect malicious apps using static features such as permissions and intents. The results of the ten classifiers are also comparable to the results obtained without using the FS technique. The results show that the FS technique is better and makes it more feasible to implement this security model on resource-constrained MDs in the MCC environment.

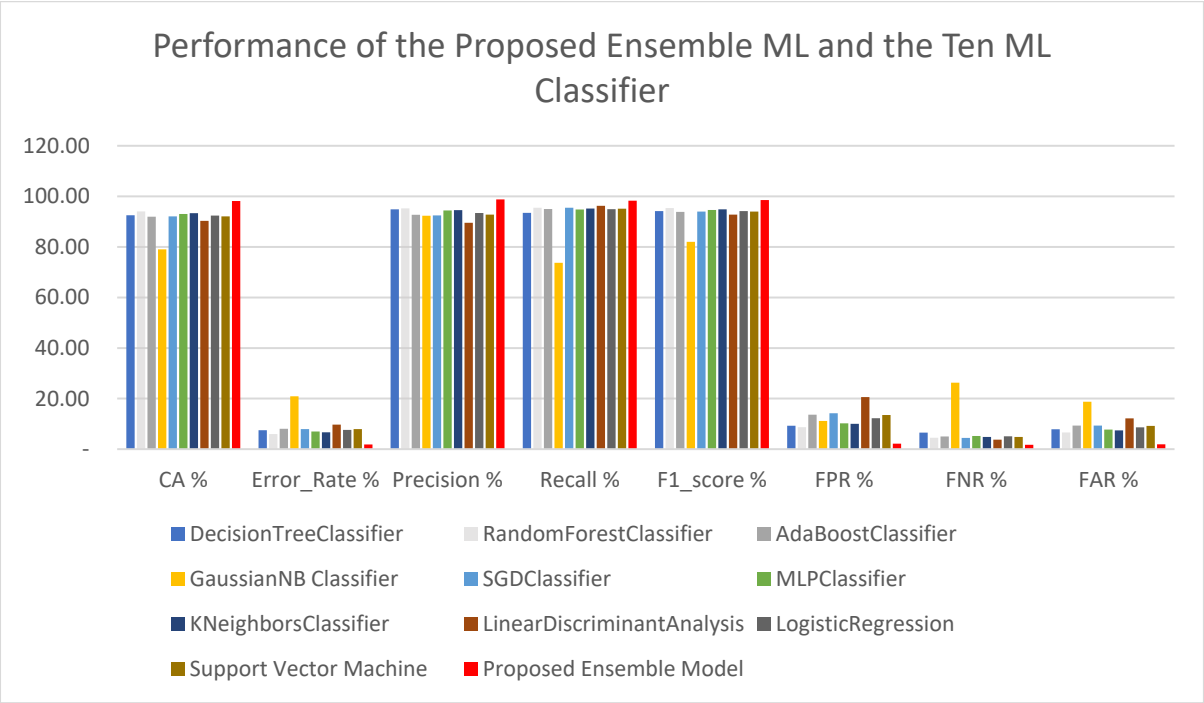

Figure 4.7 Ensemble ML Model Performance Results Using the FS Approach

The results obtained from the second experiment were validated using the ten-fold cross-validation technique as presented in Figure 4.8. The ensemble ML model achieved the highest classification accuracy of 99.01% with a minimum classification accuracy of 96.56% when validated using the ten-fold cross-validation technique. This result shows stability in the model. However, the Support Vector Machine classifier obtained a maximum accuracy of 95.05%, which was the best amongst the ten classifiers. The minimum classification accuracy of the support vector machine was 89.40, which is not too stable compared to the selected Random Forest and K-Nearest Neighbour classifiers. A more stable model with classification accuracy lies between 90.46 to 95.05 for the Neighbour classifier, while Random Forest accuracy lies between 90.78 and 94.37. The validation results show the stability and reliability of the model.

The FS approach makes it feasible to build an ML model and deploy it to resource-constrained MDs in the MCC environment because of improved detection using only 39 features compared to the entire 263 features. There is also a low computational overhead using only a few features, making this model more efficient and feasible in MDs.

As shown in Figure 4.9, the model's computational overhead in terms of the time required for both training and testing was reasonable. However, the ensemble ML model requires 3.7 seconds to test over 5,000 apps during the experiment. The high time consumption by the ensemble model resulted from the voting process amongst the three classifiers and the selection of a simple majority amongst

the individual results of the classifiers. The detection time is feasible when this model is deployed to a real-life device in the MCC environment.

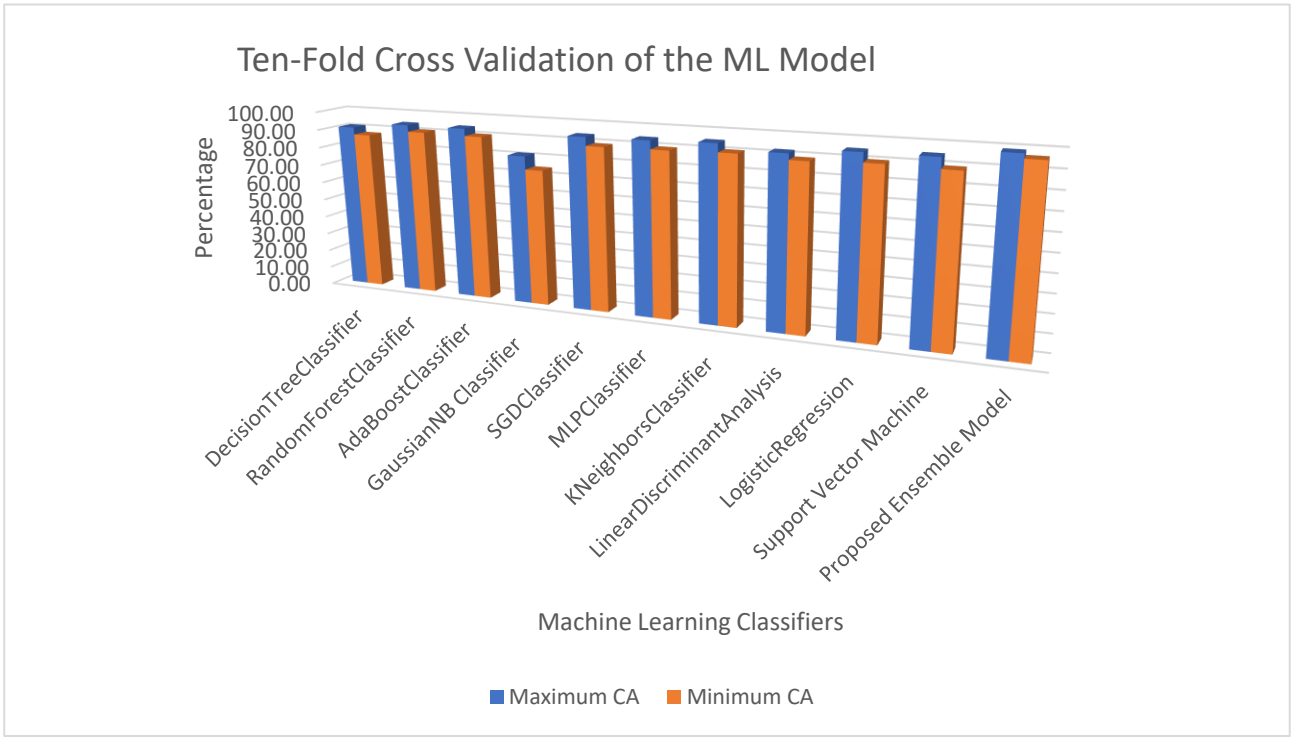

Figure 4.8 Ten-Fold Cross Validation Results of the ML Training

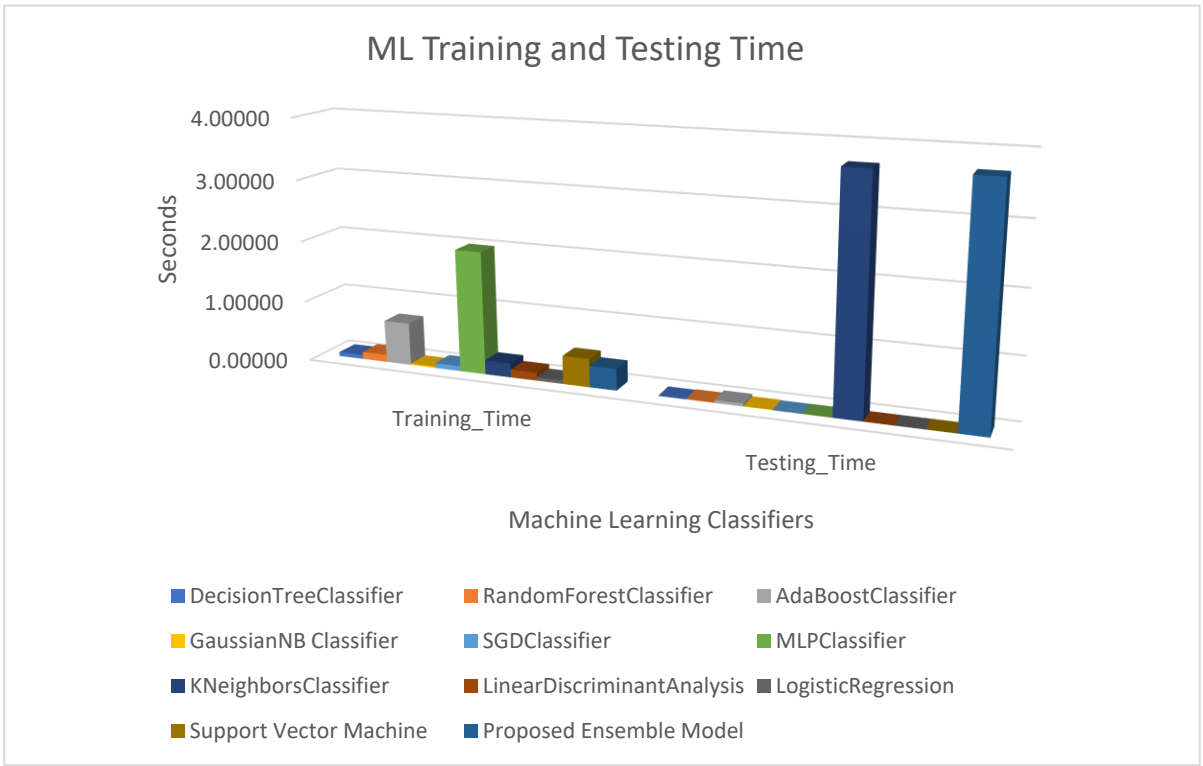

Figure 4.9 Ensemble ML Model Training and Testing Time

4.5 LABORATORY EXPERIMENT 3

This experiment aims to build an ensemble ML model that uses a dynamic analysis approach to analyze Android apps and use their behaviour to detect malicious activities at the user layer of the MCC Infrastructure.

This study collected network traffic data from 4,000 apps for two months. The selected app APK comprises 2,000 benign and 2,000 malicious samples chosen randomly from the 28,306 APK discussed in Section 4.2.1. The apps' APKs were installed on twenty different Android X Emulators (Nexus 5X) with the following hardware configuration: 1GB RAM, 512MB SD Card, 2GB internal storage, 1080 X 1920HDPI, 4 Multi-Core CPU in a remote Virtual Machine (VM) at the Auckland University of Technology, New Zealand Graduate Research Laboratory. The VM has the following hardware configuration AMD dual Core (processor) i7-8700 CPU @2.00GHz, 64GB RAM, and a 1TB hard disk drive.

A data capture app was designed to collect the network traffic from each app category (benign and malicious apps) that leveraged the Android VPN services. The data capture tool records each API request to an external service from the device emulator. The API request contains network traffic information, which includes: the number of bytes sent or received, the protocol for the request (i.e., TCP, HTTP, HTTPS, DNS, or TLS), the URL of each request, and the requested permissions and intent demanded at run-time by an app at the point of making the request. The data retrieved from each request is stored in an SQLite database for each emulator.

In each emulator in this study, 200 different apps were installed and executed. The data capture tool records each app's network traffic and the permissions and intent requests at run time to get the dynamic features of each app. The emulator captured each app's traffic data and stored it in the device database for 5 hours daily.

The emulators were left unattended for another 7 hours to record the activities of each app when the device was idle, allowing the recording of the background network calls of the apps when the user was not using the device. This process was repeated for each emulator for two days. At the end of the data acquisition period, a total of 78,285 unique network traffic records of all apps (4,000 apps) were recorded, as shown in Table 4.13

Table 4.13 Network Traffic Apps Data Distribution

Emulator	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Benign Apps	3578	3099	4329	2988	4005	3021	3566	5002	3944	2991	36523
Malicious Apps	4024	5600	5023	3456	3878	4098	5008	3207	3456	4012	41762
Total Packets	**7602**	**8699**	**9352**	**6444**	**7883**	**7119**	**8574**	**8209**	**7400**	**7003**	**78285**

4.5.1 APP DYNAMIC FEATURES EXTRACTION AND DATASET CONSTRUCTION

The dynamic features recorded in this experiment, which represent the actual app behaviour were extracted from each emulator database.

The database records of each emulator device were migrated to a Microsoft SQL Server database for further processing. The dynamic features extracted at run-time include permission, intent, and API calls to external services. The API calls to external services consist of requests made from the devices. The features of the API calls are presented in Table 4.14.

Table 4.14 API Calls Features from the App Network Traffic Data

S/N	Features	Description
1	Protocol	The type of request contains in the API call e.g., HTTP, DNS, TCP
2	Duration	The total time of the connection between the source and destination
3	Domain URL	The URL of the destination server that services the API call
4	Packet Sent	The number of packets sent from the host device(source)
5	Packet Received	The number of packets received from the destination server
6	Destination IP	The IP address of the destination server
7	Source Bytes	The total size of data sent from the host device(source)
8	Destination Bytes	The total size of data received from the destination host

The permissions and intents used in constructing dataset 5 for this experiment use the proposed filter-based FS method reported in Table 4.11 and the API calls features presented in Table 4.14 respectively using a binary vector.

Such that $PIAPI_i = \{PI_1, PI_2, PI_3, \ldots\ldots, PI_{39}, API_1, API_2, API_3\ldots\ldots, API_8\}$ where PI represent the 39 selected features (outcome of the proposed filter-based FS method) and 8 represent the total number of network features as shown in Table 4.14.

Each app APK in the fifth dataset was represented using the binary vector of permissions or intent or API requested by an app at run-time i.e., APP_i where

$$App(i)\begin{cases}1 \;\; if\; the\; permission\; or\; intent\; or\; API \;\; is\; used\; by\; the\; app\; at\; run\; time\; and\\ 0\; if\; the\; permission\; or\; intent\; API \;\; not\; used\; by\; the\; app\; at\; run\; time\end{cases}$$

The network traffic extracted as features used in constructing dataset 5 focuses only on HTTP, HTTPS, TCP, TLS, and DNS requests made from the device. These selected features of network traffic protocol are chosen because most malware usually steals sensitive information from the device and transmits the stolen data to a remote server.

4.5.2 The Proposed Ensemble ML Model Using Dynamic Analysis Approach

The proposed ensemble model uses dataset 5, constructed from the dynamic features retrieved from the apps installed on the emulator at runtime for ML training, as shown in Figure 4.10. The dynamic ensemble ML model proposed in this study focused on observing the real-time behaviours of apps' network activities by analysing the traffic data and retrieving information regarding the URL to which the traffic data is being sent. The proposed ML model was trained with an ensemble classifier using the dynamic analysis approach of bagging techniques. The choice of the bagging technique for the proposed ML model was made based on the performance results obtained during the experiment. The proposed ensemble model uses the Random Forest ML classifier as its base estimator for the training of its detection engine. The ensemble ML model was implemented in Python using the Microsoft Visual Studio 2019 IDE. The two ensemble ML models (static and dynamic feature analysis approach) proposed in this study were deployed to a cloud-based server to implement the prototype system.

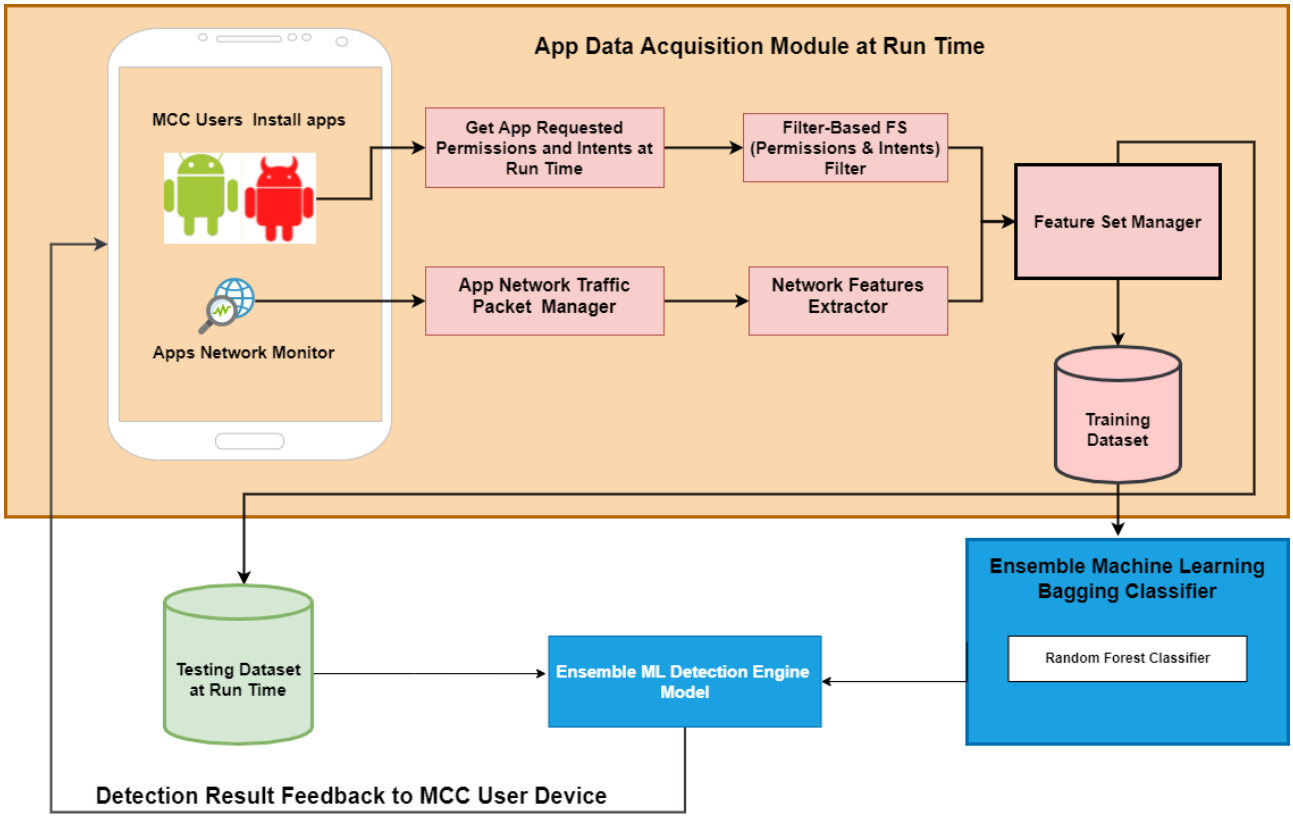

Figure 4.10 The Proposed Ensemble ML Model Using Dynamic Features Analysis Approach

4.5.3 Discussions of results Obtained from Experiment 3

The experiment reported in this section was carried out using dataset 5 constructed from permissions and intent requested by apps at run time and the network request made by the apps. The experiment uses both the voting ensembling techniques and the bagging techniques. The results obtained from the experiment also show similar performance to the previous experiments. However, the ensembling techniques using the voting techniques were outperformed by the bagging technique, hence the bagging technique for the ensemble ML model that uses the dynamic analysis approach in this study. A detailed analysis of the experiment results focuses on the evaluation of the implemented prototype system reported in chapter six of this study.

4.6 Chapter Summary

This chapter provides a detailed description of the Android OS security system. A detailed description of each of the datasets used in the experimental work is presented. The ML classification algorithms used in the study were briefly described, including the evaluation and validation metrics. The permissions and intent usage analysis was used as a basis for proposing filter-based feature selection techniques. The analysis of the experimental work reported in this chapter leads to developing two ensemble machine learning models that use both static and dynamic analysis approaches. These two models were deployed as a service to a cloud-based server for use in implementing the prototype system in the next chapter.

CHAPTER FIVE

PROTOTYPE DESIGN AND IMPLEMENTATION

The previous chapter describes the analysis of the data collected for this study. The collected data was used to construct five different datasets used in various experiments. The dataset analysis resulted in reducing the number of features to improve performance, thereby proposing a filter-based feature selection technique to reduce the number of features required to train the proposed ML models. The experimental results were evaluated, and two ensemble ML models were proposed. The proposed ML models use both static and dynamic analysis approaches of Android apps. These models were deployed as cloud-based services to implement the prototype system in this chapter.

This chapter describes the implementation of the prototype system MINDPRES. This includes a high-level view and the various subsystems that make up the prototype system. Each subsystem's implementation design details that make up the entire prototype system are also presented.

5.1 PROTOTYPE DESIGN

The prototype system design (MINDPRES) in this study focuses on the Android mobile operating environment because of its popularity, making its users prime cyber targets in the MCC environment. Data was collected and analysed to implement the prototype system as a proof of concept. The collected data is used to perform detailed ML experiments and build an ensemble ML model embedded in the prototype system. The implementation of the prototype uses a core native mobile application development environment with Java programming using the Android Studio Integrated Development Environment. The Ensemble ML Model was implemented using the Python programming language. The prototype system uses static and dynamic device behaviour analysis and ML techniques to enhance user data security at the MD node in the MCC environment. The prototype system is an improved IDPS that runs on the MD node and protects MCC resources against internal and external attacks. A description of the proposed prototype system (MINDPRES) was previously described in Chapter 3, Section 3.3.2. MINDPRES comprises three sub-system components: the Device Manager, Apps Evaluator, and the Detection Engine. A high-level view illustrating the various components in the proposed prototype system (MINDPRES) implementation is shown in Figure 5.1.

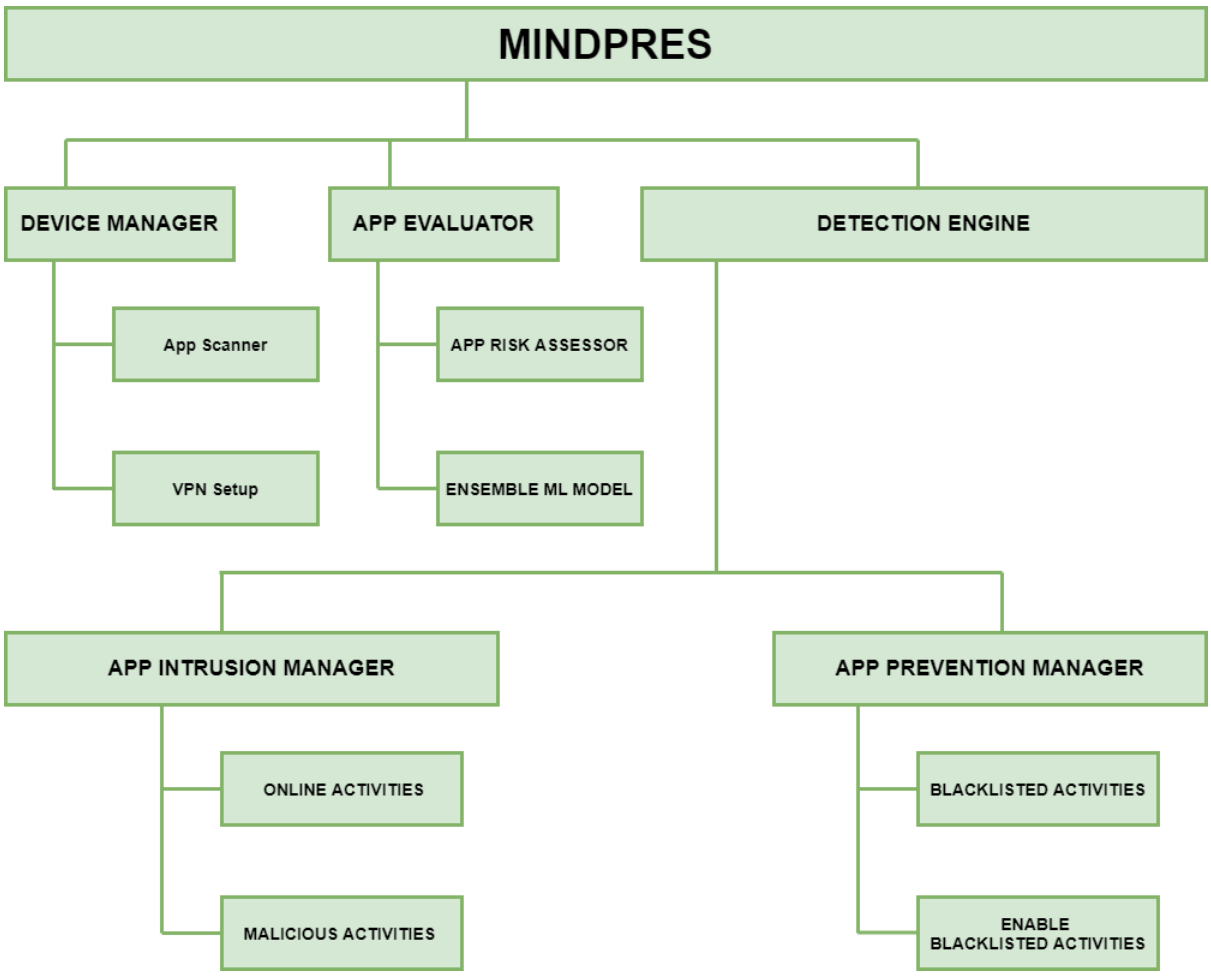

Figure 5.1 High-Level View of the Proposed Prototype System Implementation (MINDPRES)

5.1.1 THE DEVICE MANAGER

The Device Manager (DM) is a component of the prototype system as shown in Figure 5.2. The DM scans all apps that reside on the device and generate results of the total user-installed apps. The DM uses the Virtual Private Network (VPN) service libraries in the Android OS for its implementation. The VPN allows the DM to prepare the prototype system for monitoring all apps activities that reside on the devices without root-level access. The device user must grant MINDPRES access to the Android VPN service to enable the system to monitor the traffic generated by the apps in the device. The DM allows MINDPRES to run in the background while the user is using other apps and constantly monitoring the behaviour of each app in the device by analysing the network traffic generated by the apps on the device. The DM does not require any external VPN service as it only leverages the Android OS's VPN services and processes the network traffic from the apps locally on the device. Once the traffic is processed locally and stored in the device database, the DM forwards the traffic to the ensemble ML Model deployed in the cloud service end for further analysis by the detection engine for possible malicious activities.

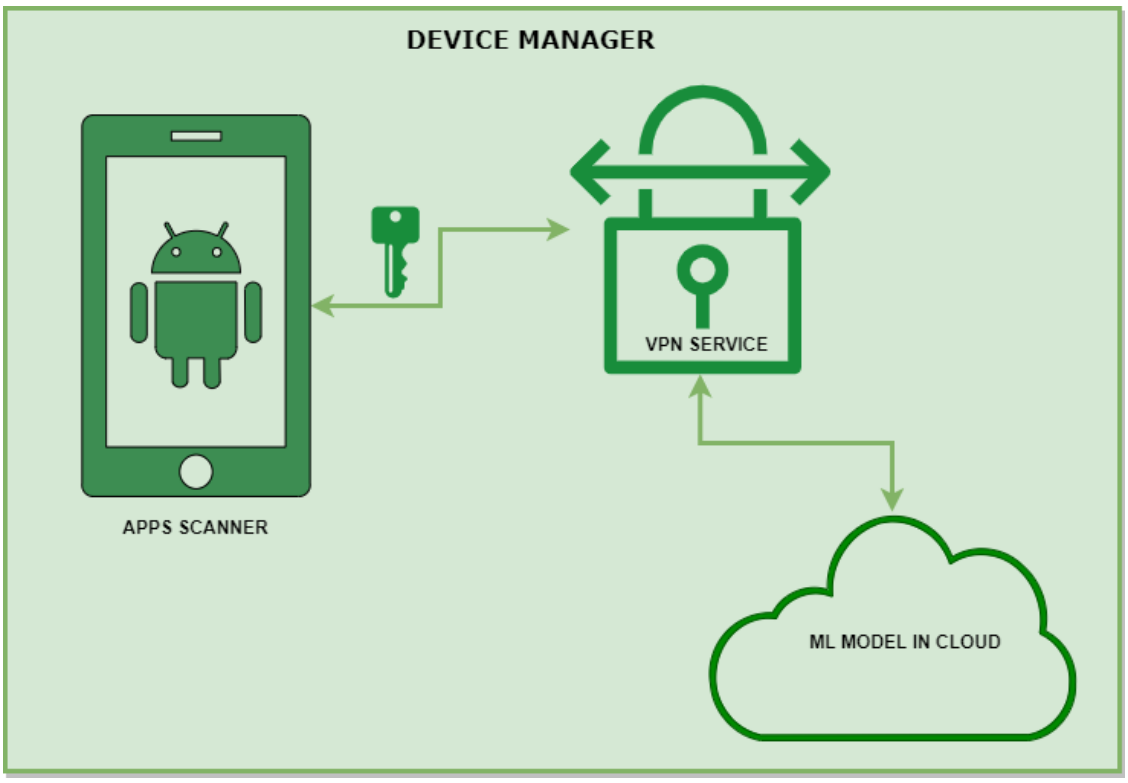

Figure 5.2 The Device Manager

5.1.2 THE APP EVALUATOR

MCC users suffer security vulnerabilities associated with apps installed on their devices, especially those malicious apps installed from the untrusted app store. To improve the security of the user layer of the MCC architecture, the app evaluator components in the proposed prototype system helps informs the MCC users of the risks associated with each app on their device.

The app evaluator displays the list of all requested permissions for each app and also provides user an option to remove unwanted apps that the app evaluator reported as risky to the security of the user data in the device. The app evaluator evaluates the risk profile of all user-installed apps and assign a risk score and category (high, medium or low) to each app. The app evaluator relies on the app's metadata information, for example the permissions and intents demanded by the app to determines its risk profile. The ensemble ML Model used by the app evaluator is deployed in an Amazon Web Service (AWS) cloud server. The model deployed in the AWS cloud server is consumed by MINDPRES using an API endpoint integrated into the app evaluator. The app evaluator requires some selected permissions and intent using the ensemble ML model via the Permission and Intent Filter. These features (permissions and intents) are passed as parameters to the API endpoint, and a request is sent to the ensemble model in the cloud server. The cloud server processes the request and sends feedback to the devices that make the request. The feedback sent

to the devices is either zero or one. The value of zero indicates that the ensemble ML model has classified the app that demanded the permissions and intent as a benign app, while a value of one indicates that the app is classified as malicious.

The output of the app evaluator depends on fuzzy classification rules proposed in this study. The fuzzy sets include the classification outcome of the ensemble ML model using the static analysis approach discussed in Chapter Four, Section 4.4.3 and the score generated by the Risk Assessor in the prototype system using the probabilistic approach discussed below in this section to determine the risk category of the app (i.e., High, Medium, or Low) as shown in Figure 5.3. For example, an app classified as benign by the ensemble ML model is categorized as a high-risk app if and only if the risk value generated by the risk assessor using the probability approach proposed in this study is between 0.86 and 1.00. Also, an app classified as benign by the ensemble ML model is categorized as medium risk if the risk value is between 0.66 and 0.85, or low risk if the risk value is below 0.00 and 0.65.

In addition, an app classified as malicious by the ensemble ML model is categorized as a high-risk app if and only if the risk value generated by the risk assessor using the probability approach proposed in this study is between 0.61 and 1.00. Also, an app classified as malicious by the ensemble ML model is categorized as either medium-risk if the risk value is between 0.26 and 0.60 or low-risk if the risk value is below 0.00 and 0.25.

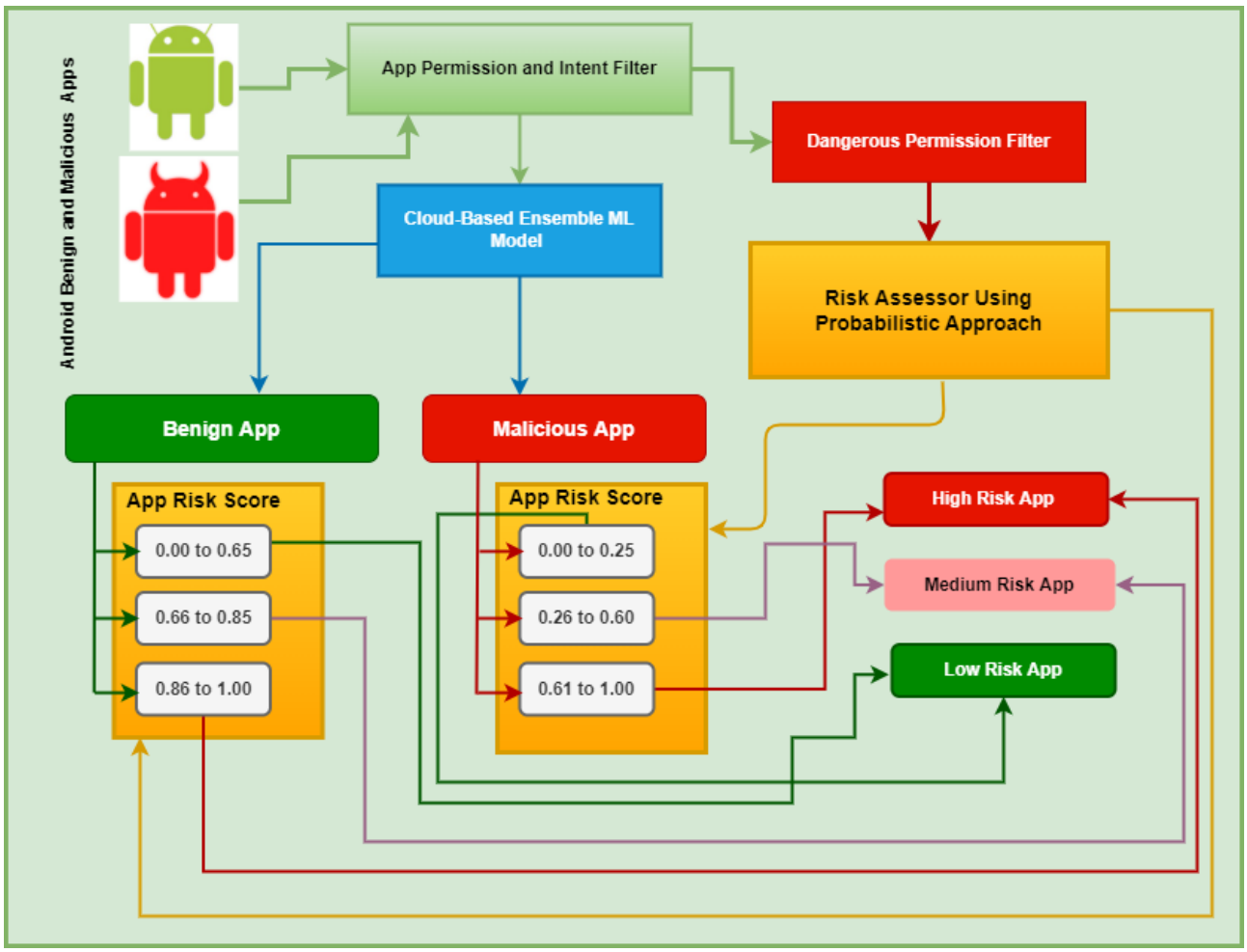

Figure 5.3 The App Evaluator

The probabilistic risk assessor model proposed in this study determines the risk score of the app based on how much; the app uses these fifteen dangerous permissions (P1 to P15) as discussed in Chapter Four, Section 4.2.2, Table 4.5. The choice of these sets of permissions relies on the information provided in the Android OS documentation. This set of dangerous permissions, when granted to an app, poses a greater risk to the security and privacy of user data stored on the device. The greater risk posed by these sets of permissions allows the app evaluator to determine the risk score of an app by its usage patterns. The equations (5.1 to 5. 4) are used to define the proposed risk function (equations 5.5 and 5.6) used for the implementation of the risk assessment module of the proposed MINDPRES system (equations 5.7 and 5.8). The choice of this approach (a probability risk value function that is based on the statistics of the data collected in this study) is supported by previous studies that apply similar methods; for example, in determining the risk value of Android apps (Peng et al, 2012; Cen et al, 2014; Mat et al, 2022).

The probabilistic risk assessment model proposed in this study and used by the app evaluator is described as follows:

Let $S = \{P_1, P_2, P_3, \dots, P_n\}$ be a non-empty set containing all known dangerous permissions in the dataset, P_i, such that the cardinality of the set S, $|S| = n < \infty$ (i.e., S is a finite set and n=15).

For each dangerous permission, P_i, $1 \leq i \leq n$, let α_i be the ratio of usage of the permission P_i by the total number of malicious apps in the dataset discussed in Chapter Four, Sections 4.2 and 4.2.2, Tables 4.1 and 4.5. That is,

$$\alpha_i = \frac{Total\ number\ of\ malicious\ apps\ that\ use\ P_i}{Total\ number\ of\ malicious\ apps\ in\ dataset} \qquad Eq.5.1$$

Similarly, let β_i be the ratio of usage of the permission P_i by benign applications to the entire benign data set as discussed in Chapter Four, Sections 4.2 and 4.2.2, Tables 4.1 and 4.5. That is,

$$\beta_i = \frac{Total\ number\ of\ benign\ apps\ that\ use\ P_i}{Total\ number\ of\ benign\ apps\ in\ dataset} \qquad Eq.5.2$$

In this case, both α_i and β_i can take values ranging from 0 to 1 only, for all values of i. (i.e., $0 \leq \alpha_i, \beta_i \leq 1$, for all i).

Corresponding to each permission P_i, define the function $\lambda(i)$ by

$$\lambda(i) = \begin{cases} 1, if\ dangerous\ permission\ is\ used\ by\ the\ app. \\ 0, if\ dangerous\ permission\ is\ not\ used\ by\ the\ app. \end{cases} \qquad Eq.5.3$$

Next, the value of the constant k, is set for every app to be tested, as

$$k = \sum_{i=1}^{n} \lambda(i) \qquad Eq.5.4$$

Hence, k is a positive integer such that $0 \leq k \leq n$.

Finally, the risk function $R(\lambda(i), \alpha_i, \beta_i)$ of an Android app X is defined by

$$R(\lambda(i), \alpha_i, \beta_i) = \frac{1}{k} \sum_{i=1}^{n} \lambda(i) \sqrt{\alpha_i - \alpha_i \beta_i} \qquad Eq.5.5$$

These evaluates the risk of all dangerous permissions requested by app X. Thus, $R(\lambda(i), \alpha_i, \beta_i)$ is a probability valued function that determine the risk value of each dangerous permission such that R lies in the interval $[0,1]$ (i.e., $0 \leq R(\lambda(i), \alpha_i, \beta_i) \leq 1$), for each Android app to be evaluated. Suppose an app uses any of these dangerous permissions the risk value generated by the proposed model will be greater than zero but less than or equal to 1.

Considering the risk function $R(\lambda(i), \alpha_i, \beta_i)$ in $(Eq5.5)$ above, it follows straightforward that for an app X with $\alpha_i = 0$ (which implies that the dangerous permission P_i was never requested or demanded by all malicious app), for all $\lambda(i) = 1$ (i.e. for all dangerous permissions requested in the app X to be evaluated), then the risk function generates

$$R = 0 \qquad Eq.5.6$$

Clearly, it is evident that independent of β_i (ratio of the dangerous permission P_i requested by benign apps as discussed in Chapter Four, Sections 4.2 and 4.2.2, Tables 4.1 and 4.5, the risk function $R = 0$, whenever $\alpha_i = 0$, for every permission P_i found in app X.

In Figure 5.3, the app evaluator, analysed each app in the device by sending a request to the ensemble ML model deployed in the cloud server. The request contains the selected features require by the model that consist of a set of permissions and intent demanded by each app and the proposed probabilistic value function score of the list of dangerous permission demanded by the app as discussed in this section. The app evaluator uses a set of fuzzy rules define on the risk score output of equation Eq. 5.5 as a membership function to categorize each app into one of the following three (3) distinct groups namely as Low-risk app, Medium-risk app, or High-risk app.
This study constructs the following fuzzy rules with the help of (AND) logical operators to determine the risk category of each app by the app evaluator.

Rule 1: (if app risk score >=0.0 ***AND*** risk score <=0.65) ***AND*** ensemble ML classification of the app is **benign** THEN the app is classifies as a **low risk app**

Rule 2: If (app risk score >0.65 ***AND*** risk score <=0.85) ***AND*** ensemble ML classification of the app is **benign** THEN the app is classifies as a **medium risk app**

Rule 3: If (app risk score >0.85 ***AND*** risk score <=1.00) ***AND*** ensemble ML classification of the app is **benign** THEN the app is classifies as a **high risk app**

Rule 4: If (app risk score >=0.00 ***AND*** risk score <=0.25) ***AND*** ensemble ML classification of the app is **malicious** THEN the app is classifies as a **low risk app**

Rule 5: If (app risk score >0.25 ***AND*** risk score <=0.60) ***AND*** ensemble ML classification of the app is **malicious** THEN the app is classifies as a **medium risk app**

Rule 6: If (app risk score >0.60 ***AND*** risk score <=1.00) ***AND*** ensemble ML classification of the app is **malicious** THEN the app is classifies as a **high risk app**

These fuzzy set rules are applied on the following possible case scenarios.

Case A: Low risk app: Suppose $\alpha_i = 0$ for each dangerous permission P_i found in an app X, $R(\lambda(i), \alpha_i, \beta_i) = R_{min} = 0$, which implies the lowest risk irrespective of the outcome of the ensemble ML classification such app is classify as a Low-risk app.

Specifically, if for every dangerous permission P_i requested in an app X, $\alpha_i < \beta_i$ (i.e. α_i is much less than β_i), then $R(\lambda(i), \alpha_i, \beta_i) \in [0,0.25]$ (i.e. $0 \leq R \leq 0.25$) is said to be a low-risk app irrespective of the outcome of the ensemble ML classification. The risk classification for this scenario use either rule 1 or rule 4 depending on the outcome of ensemble ML classification model.

In addition, using rule 1, an app can also be considered as low-risk app if and only if the result of the ensemble ML classification is benign and the risk lies between 0.00 to 0.65($R(\lambda(i), \alpha_i, \beta_i) \in [0,0.65]$ (i.e., $0 \leq R \leq 0.65$))

Case B: Medium risk app: In this scenario, an app is considered a medium risk depending on the result of the ensemble ML classification and the region the risk value score of the app falls into. Using rule 2, suppose the ensemble ML model classified the app X as benign and for every dangerous permission P_i found in app X with risk function $R(\lambda(i), \alpha_i, \beta_i) \in (0.66,0.85]$ (i.e., $0.66 < R \leq 0.85$) is said to be medium-risk app.

Similarly, using rule 5, suppose the ensemble ML model classified the app X as malicious and for every dangerous permission P_i found in app X with risk function $R(\lambda(i), \alpha_i, \beta_i) \in (0.26,0.60]$ (i.e., $0.26 < R \leq 0.60$) is said to be medium-risk app.

Case C: High risk app: Using rule 3 or 6, suppose that $\alpha_i = \beta_i \leq 0.50$, for every dangerous permission P_i found in an app X, then $R(\lambda(i), \alpha_i, \beta_i) \leq 0.50$. (This implies that the dangerous permission P_i is requested at the same rate in both malicious and benign applications, and of which rate is less than or exactly 50%). But supposing for each and every dangerous permission P_i found in any arbitrary app X, $\alpha_i = 1$ and $\beta_i = 0$ (i.e. if all the permissions $P_i's$ are requested for at 100% rate in malicious applications and 0% rate in benign applications), then $R(\lambda(i), \alpha_i, \beta_i) = R_{max} = 1$. which implies the highest risk irrespective of the outcome of the ensemble ML model classification such app is classify as a High-risk app.

Suppose the ensemble ML model classified app X as benign and for every dangerous permission P_i found in app X with risk function $R(\lambda(i), \alpha_i, \beta_i) \in (0.86,1.00]$ (i.e., $0.86 < R \leq 1.00$) is said to be high-risk app.

Similarly, suppose the ensemble ML model classified app X as malicious and for every dangerous permission P_i found in app X with risk function $R(\lambda(i), \alpha_i, \beta_i) \in (0.61,1.00]$ (i.e., $0.61 < R \leq 1.00$) is said to be high-risk app.

In general, the boundaries used for risk categorization of apps in this study is based on the analysis of the dangerous permissions usage by both malicious and benign apps and the results of the ensemble ML classification model. The app evaluator in MINDPRES as shown in Figure 5.3 classify an Android app by analysing each dangerous permission P_i requested therein, the following is obtainable

if the results of the ensemble ML Prediction for an app is Benign:

$$R(\lambda(i), \alpha_i, \beta_i) = \begin{cases} x \in [0,0.65], for\ low\ risk\ app \\ x \in [0.66,0.85], for\ medium\ risk\ app \\ x \in [0.86,1.00], for\ high\ risk\ app \end{cases} \quad Eq.\ 5.7$$

where $x \in R$, the set of real numbers that denotes the risk score of the app.

and if the results of the ensemble ML Prediction for an app is Malicious:

$$R(\lambda(i), \alpha_i, \beta_i) = \begin{cases} x \in [0,0.25], for\ low\ risk\ app \\ x \in [0.26,0.60], for\ medium\ risk\ app \\ x \in \big([0.61,1.00]\big], for\ high\ risk\ app \end{cases} \quad Eq.\ 5.8$$

where $x \in R$, the set of real numbers that denotes the risk score of the app.

5.1.3 THE DETECTION ENGINE

The detection engine is the security component of the prototype system (MINDPRES) that monitors apps' behaviours in real-time. The detection engine comprises two main sub-systems, shown in Figure 5.1: the app intrusion manager and the app prevention manager.

A. The App Intrusion Manager: The app intrusion manager uses a host-based IDS approach to analyze the behaviour of all apps that reside on the devices, as shown in Figure 5.4. The app intrusion manager works in conjunction with the device manager by leveraging the Android VPN services to monitor the behaviours of all apps on the device. The app intrusion manager applies a dynamic analysis approach to analyze the behaviour of apps that reside locally on the device using the proposed ensemble ML model discussed in Chapter Four, Section 4.5.2.

The intrusion manager analyses the permissions and intents requested by an app and extracts the selected features (permissions and intents requested at run-time). The permission and intent extractor of the intrusion manager analyses the actual permission and intent requested by an app at run time rather than the permission and intent declared in the

app manifest file. These enable the intrusion manager to know the actual resources that the app is accessing at any point in time.

In addition, the app evaluator uses all permissions and intents declared by the developers that the app intends to use during the actual execution of the app. In contrast, the intrusion manager analyses only permissions and intents requested during the app's execution. The ability to analyse the actual permissions and intent enables the intrusion manager to predict the actual behaviour rather than the intended behaviour of the app at run-time. The intrusion manager only selects the permissions and intent to belong to the list of selected features used in the training of the ensemble model as described in Chapter Four, Section 4.4.2, Table 4.11. An app demands the selected permissions and intents at run-time, and the extracted network traffic of the request is passed to the feature set manager. The feature input received by the feature set manager is forwarded to the ensemble ML deployed in the AWS cloud service for analysis. The classification results produced by the ensemble ML model (malicious or benign request) are forwarded to the intrusion assessor for an investigation to determine if the request is malicious or not.

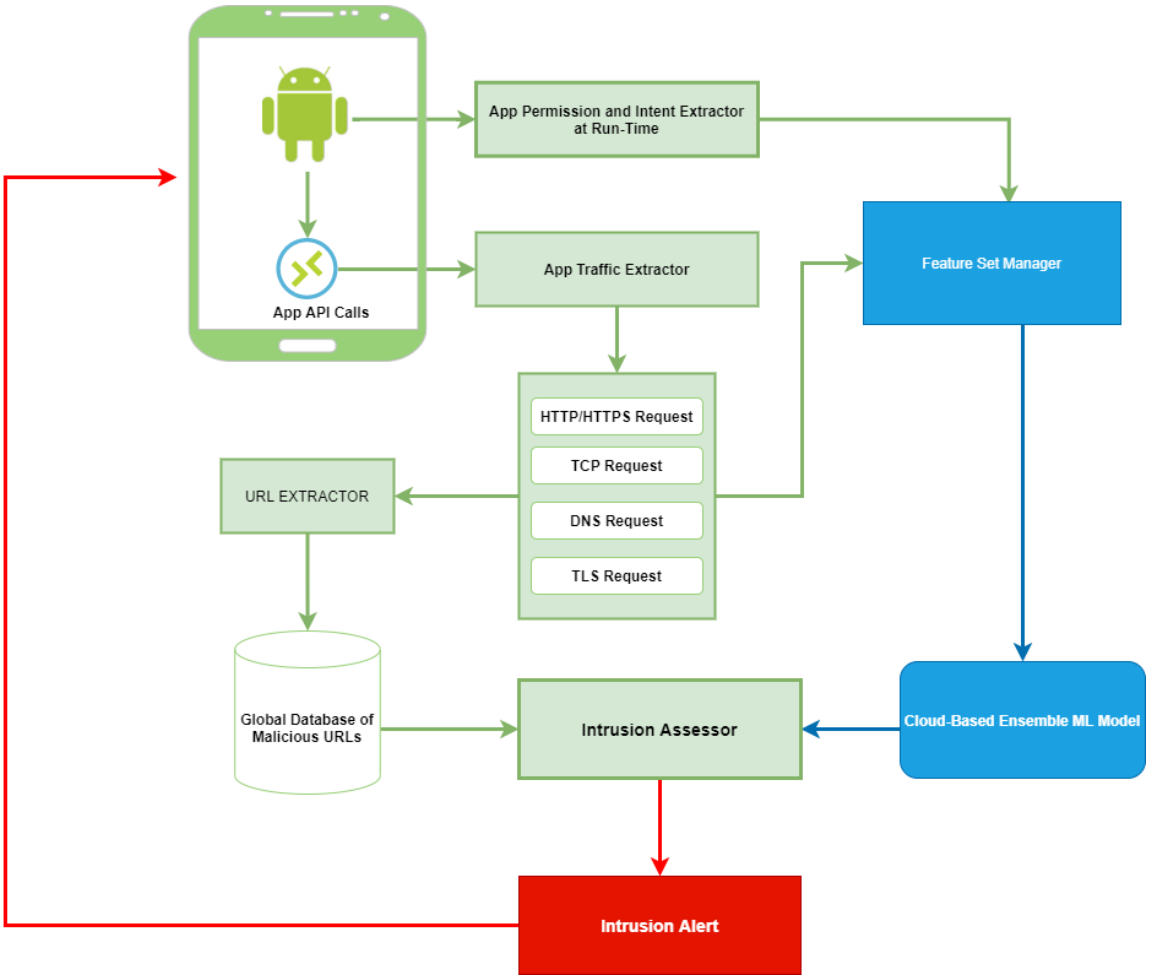

Figure 5.4 App Intrusion Manager

The app traffic extractor in the intrusion manager captures all network traffic from all apps on the device by analysing the actual API calls, consisting of HTTP/HTTPS requests, TCP requests, TLS requests, and DNS requests. The network traffic features (such as the request protocol, the duration of the connection, the number of bytes sent from the device, the number of bytes received from the destination host, the packet size sent, the packet size received from the destination host, the IP address of the destination request, and the source request) are forwarded to the feature set manager for further processing. The app traffic extractor also extracts the URL from the API calls and checks it against a known global malicious URL database to determine if the API URL call is malicious or not. The results from the global database of known malicious URLs are grouped into four categories: malware, spamming, phishing, and suspicious URLs. The results from the API request to analyze the URL from the global database are forwarded to the intrusion assessor for the final assessment.

The intrusion assessor analyses the results obtained from the global database of malicious URLs and the cloud-based ensemble ML model to determine if app traffic contains a URL that has been classified as malicious or not. The final results by the intrusion assessor show that a specific packet might be suspicious by analysing the abnormality in the packet, checking the duration of connection and the packet size information combined with the results of the global database of malicious URLs. The intrusion assessor flags any network traffic or packets obtained as malicious if and only if the cloud-based ensemble ML model predicts it as malicious or if the global malicious URL results return true for any of the four categories, namely malware URL, suspicious phishing URL, or spamming URL. The URL is marked as "red," and an intrusion alert is sent back as feedback to the intrusion manager component of the detection engine.

B. The App Prevention Manager: As shown in Figure 5.1, the prevention manager is a detection engine's mitigation component that checks for intrusion alerts reported by the intrusion assessor. The prevention manager identifies the app that requests by checking the traffic data and communicating with the device manager, leveraging the VPN services, and automatically blocking any traffic coming from the app to the reported destination IP address. The prevention manager shows the list of backlisted traffic and allows the user to enable the app to call the blacklisted URL based on the MD user assessment.

5.2 PROTOTYPE IMPLEMENTATION

The implementation of the prototype system in this study applies the object-oriented analysis and design (OOAD) paradigm. This approach provides flexibility in designing artifacts for use in a real-life scenario.

5.2.1 THE DATABASE DESIGN

The prototype system's database was implemented using an open-source database system named SQLite. This database is available for all Android devices. The database stores the traffic data captured on the device, and the intrusion manager will analyze each piece of data in the background for possible intrusions. The Entity-Relationship (ER) of the database tables and how they are used to implement the prototype system (MINDPRES) is shown in Figure 5.5.

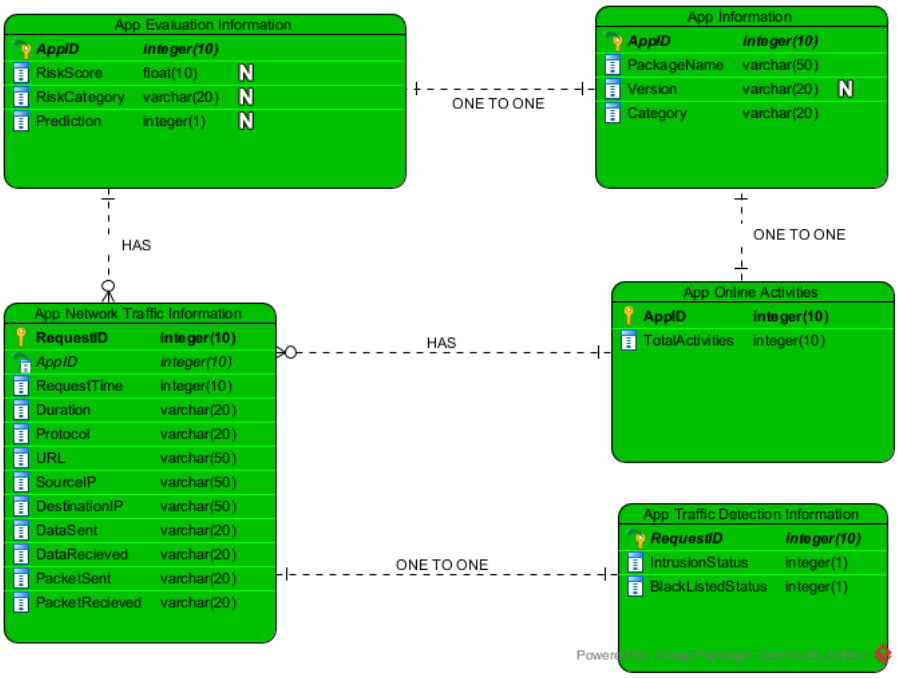

Figure 5.5 MINDPRES-Database Entity Relationship Diagram

A. The App Information Table*:* The app information table structure is shown in Table 5.1. This table stores information on all apps that reside on the devices. The app information table is populated by the device manager based on its analysis of all apps that reside on the device.

Table 5.1 App Information Table

Column Name	Description	Data Type
AppID **(PK)**	This column holds a unique identity for each app.	Integer (10)
Package Name	This column holds the package name of each app.	Varchar (50)
Version	This column holds the version details of each app	Varchar (20)
Category	This column holds the category information of each app as either user install apps or system apps	Varchar (20)

B. The App Evaluation Table: The structure of the app evaluation table is shown in Table 5.2. This table stores both the results of the app evaluator and the ensemble ML prediction of each user-installed app using the metadata information (permissions and intents).

Table 5.2 App Evaluation Table

Column Name	Description	Data Type
AppID **(PK)**	This column holds a unique identity for each app.	Integer (10)
RiskScore	This column holds the risk score of each app.	Float (10)
RiskCategory	This column holds the risk category of each app	Varchar (20)
Prediction	This column holds the ML Prediction result for each app	Integer (1)

C. The App Network Traffic Table: The structure of the app network traffic table is shown in Table 5.3. This table stores information retrieved from each request made by an app that passes through the VPN. The table also stores details of each connection request via an API call to an external service.

Table 5.3 App Network Traffic Table

Column Name	Description	Data Type
RequestID **(PK)**	This column holds a unique identity for each app network traffic connection made from the device	int
AppID **(FK)**	This column holds a unique identity for each app.	int
RequestTime	This column holds the date and time the request was initiated	datetime
Duration	This column holds the time interval for each API call request.	int
Protocol	This column holds the request type information	Varchar (20)
URL	This column holds the URL information of each request	Varchar (50)
SourceIP	This column holds the IP address of the source request	Varchar (50)
DestinationIP	This column holds the IP address of the destination of the API call request	Varchar (50)
DataSent	This column holds size of the data sent	Varchar (20)
DataRecieved	This column holds size of the data received	Varchar (20)
PacketSent	This column holds number of the packet sent	Varchar (20)
PacketRecieved	This column holds number of the packet received	Varchar (20)

D. The App Online Activities Table: The structure of the app's online activities table is shown in Table 5.4. This table stores information on all apps and the total number of all online activities.

Table 5.4 App Online Activities Table

Column Name	Description	Data Type
AppID **(PK)**	This column holds a unique identity for each app.	int
TotalActivities	This column holds the total number of online activities for each app	int

E. The App Traffic Detection Table: The structure of the app traffic detection table is shown in Table 5.5. This table stores information about the detection and prevention engine results for each network traffic.

Table 5.5 App Traffic Detection Table

Column Name	Description	Data Type
RequestID **(PK)**	This column holds a unique identity for each app.	int
IntrusionStatus	This column holds the intrusion result status of each online activities request made by each app.	int
BlacklistedStatus	This column holds the backlisted status of each online activities request made by each app.	int

5.2.2 THE IMPLEMENTATION TOOLS

The proposed ensemble ML models used to implement the prototype system in this study were implemented using Python programming language. The robust ML libraries that is supported in python programming language makes it's a preferred tool for the design and development of ensemble ML models in this study. This study uses the python's scikit-learn, pandas, and NumPy libraries to develop the ensemble ML model. The ensemble ML models used to implement the prototype system (MINDPRES) were also deployed to the AWS cloud container. MINDPRES utilized the corresponding endpoint API implementation of the ensemble ML models for app evaluation and intrusion detection of malicious activities of suspicious API calls from the device.

The prototype system was implemented using the Android Studio Integrated Development Environment (IDE). The Android Studio IDE was used to design the mobile app on the MCC user's end. The language used for implementing the mobile app is the programming language. Using the Java programming language to develop the prototype system was chosen because of its efficiency in developing native mobile apps that run on the device. This study's prototype system requirements require a programming language that provides an efficient library system that enables access to the device's low-level resources. MINDPRES requires some native access to the device's resources, such as the VPN service and the package manager. Java provides efficient libraries that provide an efficient way to communicate with the Android OS kernel. The libraries supported by the Java programming language enable the prototype to efficiently analyze all the apps on the devices and provide a way to intercept all API calls from each app that resides on the device.

The User Interface (UI) was designed using the Extended Markup Language (XML) in the Android Studio. The prototype implementation used the Android Emulators during the development phase and was later tested on various real-life Android devices. The UIs designed for the implementation of the prototype system are discussed as follows:

A. ***The Device Manager UI:*** This is the first UI displayed to the app's end-user when the app is launched. The device manager UI has a different process that runs in the background. This UI provides greater functionality and asks the user to accept the VPN service for MINDPRES to monitor the activities of all apps on the device. The device manager UI also displayed the total number of users who had installed apps on the device, as shown in Figure 5.6.

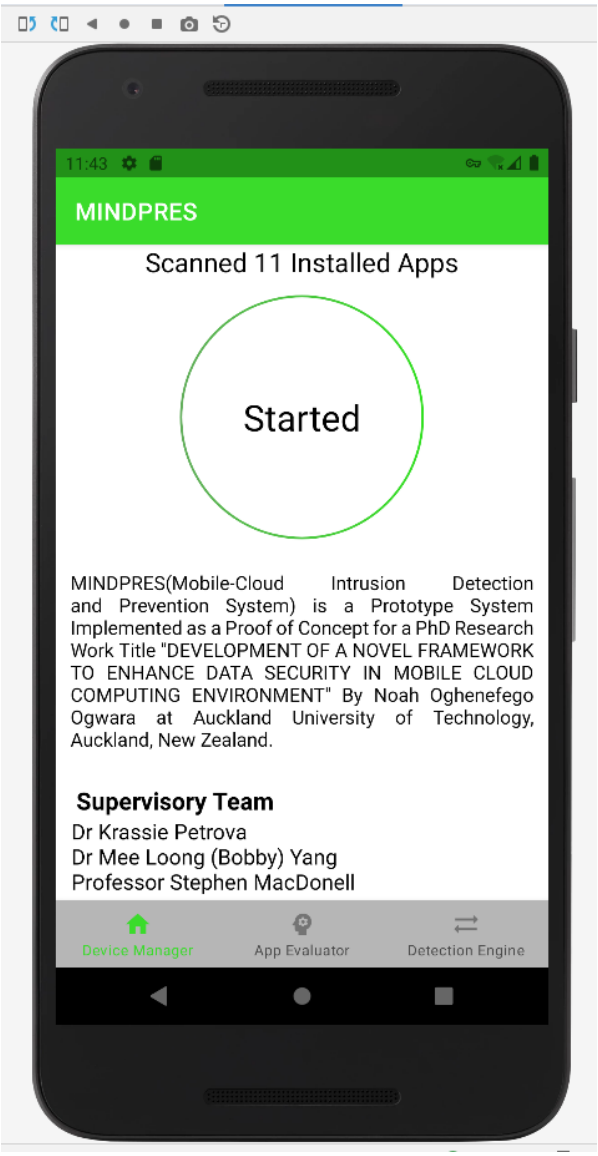

Figure 5.6 MINDPRES-Device Manager UI

B. ***The App Evaluator UI:*** The app evaluator UI evaluates all user-installed apps by analysing the metadata features (permissions and intents) demanded by the apps. The metadata evaluation results are used to generate the risk score and category for each app by combining the results of the ensemble ML model with the probability risk value function as discussed in Section 5.1.2 of this chapter. The app UI also shows the list of permissions demanded by an app and allows the user to uninstall apps that pose more risk to the device, as shown in Figure 5.7.

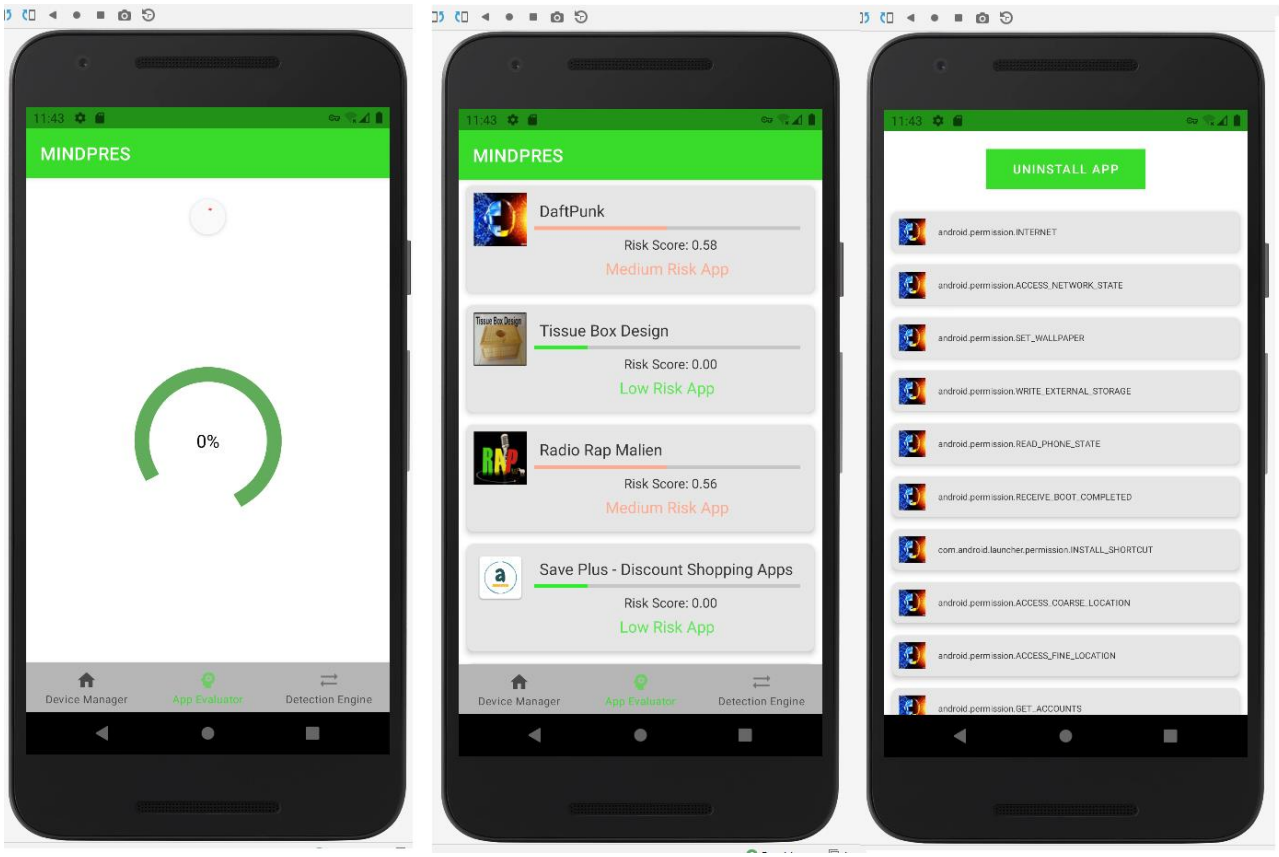

Figure 5.7 MINDPRES-App Evaluator UI

C. The Detection Engine UI: The detection engine UI contains three different tabs embedded in the UI. The first tab displays the list of all apps, the total number of online requests made by each app, and MINDPRES monitors that. The UI also provides a view containing the list of API calls made to external URLs from the device and the details of the API calls made. The second tab displays the list of apps with the total number of malicious API calls detected by the intrusion assessor. This tab also shows the API call URL that is flagged as malicious. The third tab displays the list of all blacklisted API calls and allows the user to either enable or disable a blacklisted URL, as shown in Figures 5.8 and 5.9.

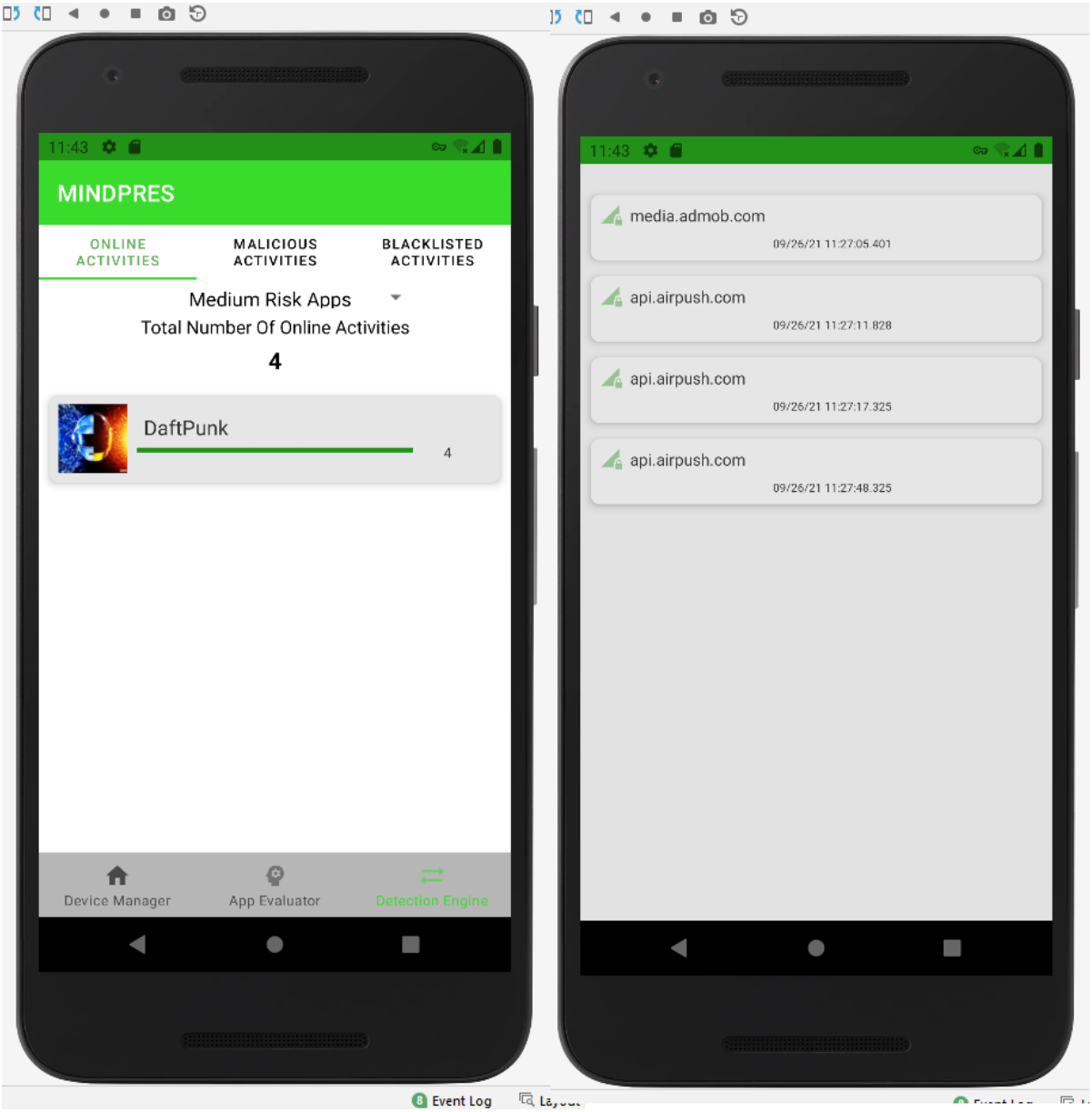

Figure 5.8 MINDPRES-App Detection Engine-Online Activities Tab UI

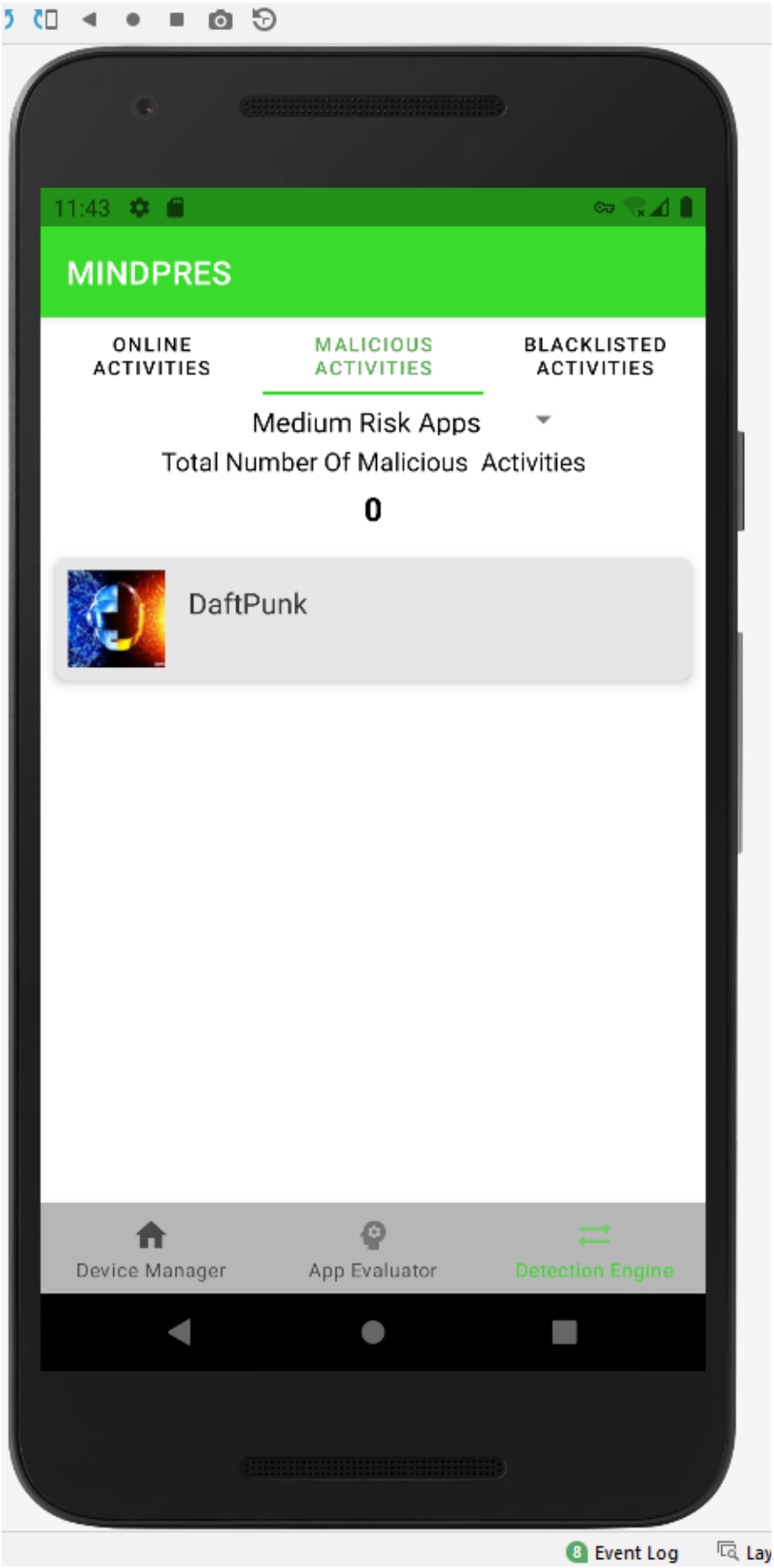

Figure 5.9 MINDPRES-App Detection Engine-Malicious Activities Tab UI

5.2.3 THE UNIFIED MODELLING OF THE PROTOTYPE SYSTEM

This study uses the Unified Modelling Language (UML) approach to show the internal structures of the various objects and how they interact with each other in the implemented prototype system. The UML diagrams used for the prototype implementation are discussed as follows:

A. ***The Class Diagram of the Prototype System:*** The class diagram is a structural type of UML diagram that shows the internal structures of the various classes that make up the prototype system and how each class interacts with the other. The class diagrams also show each class's different attributes and the associated methods required to function. In this study, the main activity class is the backbone of the prototype system. The main_activity class has three classes that are associated with it. These classes include the device_manager class, app_evaluator class, and detection_engine class. The three classes use the various subclasses for their functionality, as shown in the class diagram presented in Figure 1 in appendix B.

   The setAppStateListener is a setter method that sets the value of the m_Listener attributes of the main_activity class. Similarly, the getAppUri methods set the value of the URL retrieved by each app to the m_uri attributes of the main_activity class. The check_permission method evaluates the device's VPN permissions to allow the prototype system to either capture app activities or not. In addition, the captureServiceOk and captureServiceResults are responsible for capturing all API calls from all apps that reside on the device.

   The device_manager class directly depends on the main_activity class for some of its basic functionality. The OnCreate method of the device_manager class initiates the class activity and loads all the necessary data passed on by the main_activity class. The device_manager class sets up the VPN connection for the devices and gets the device app list returned by the getAppList method. The device_manager class is also responsible for evaluating the total number of user-installed apps on the devices. It returns the total number of apps and device information via the getAppsCount and getDeviceID methods.

   The app_evaluator class depends also on the main_activity class to process the state of each apps that resides on the device. The app_evaluator class inherits the methods in AllAppData class and the permissionList class. The AllAppData class also depends directly on the AppRiskModel class and AppRiskData class. The app_evaluator has five attributes that depends on the methods define within its entity. The app_evaluator class executes the getRiskCategory of all apps which depends on the results return by the getMLPrediction methods and getRiskScore method that depends directly on other subclass associated with the app_evaluator class and set the app risk category for all users install apps in the device.

The detection_engine class depends on the main_activity class to get result of activities and URL request that was capture. The detection_engine class also inherits the AppOnlineActivities class that depend on the AllAppData class and AppActivites class. This enables the detection engine to execute the getAppOnlineActivites method that set the values of Apps, TrafficData and TotalAppActivites attributes of the AppOnlineActivities class. These enable the detection engines to monitor the activities of all apps in the device.

The GetAllMaliciousActivites of the detection engines depends on the AppMaliciousActivities class that inherits methods in the AppActivites class that depends directly on the URL_Extractor class to set the maliciousActivites attribute of the detection engine. Similarly, the GetAllBlacklistedActivites method of the detection engines also inherits methods of AppActivites class and the AppBlacklistedActivites class to set the attributes values of blacklistedActivites of the detection engine class. The detection engine also executes the enableBlacklistedActivites of AppBlacklistedActivites class to allow a blacklisted activities to be active in the device.

B. The Sequence Diagram of the Prototype System: The sequence diagram is behavioural type of UML diagram that shows the behavioural of the objects in the prototype system. The sequence diagrams show the interaction activities amongst the various objects that make up the prototype system, as shown in Figure 2 in appendix B.

The sequence diagram depicts the activities sequence on how the system interact with each of the object. First, the user launches the prototype system the system gets the information of the device and prompt the user to grant the prototype system access to the VPN service which enable the system to monitor the activities of all apps that resides on the device. The system will not monitor the activities of the apps if the user denies system access to the VPN services.

Secondly, whenever access to the VPN services is granted by the device user, the prototype system automatically setup its own VPN connection that enable MINDPRES to monitor all the activities of all the apps that resides on the device. The system also scanned all user installed apps that resides on the device and returns the total numbers of apps that resides on the device.

Thirdly, the App evaluator extract the permission demanded by each apps that resides on the devices and used the information to evaluate the riskiness of each app and returns the result of the risk assessment by combining the result of the ML prediction and risk score assessor model of the prototype system

Finally, the detection engine works in parallel with the device manager once the VPN connection has been established the prototype system start capturing all activities of all apps within the device. These activities captured are also further examined for possible intrusions and filter out activities that are predicted as malicious by the detection engine and automatically block all malicious activities. The system also allows the user to activate block activities in the event of false alarm by the system.

C. **The Activity Diagram of the Prototype System:** The activity diagram is behavioural type of UML diagram that shows the dynamic aspects of the prototype system. The activity diagram models the flow between activities within a system. The activity diagram shows the coordination of activities to provide services at different levels of abstraction within the system abstraction. The activity diagrams show different events and their associated operations required to complete them. In this study, the UML diagram presented in Figure 3 in appendix B, shows how the various activities flow from one point to another. It shows an abstraction of events between the MD user and the prototype system (MINDPRES).

First, the user launches the prototype system and initiates the VPN connection that requires the user's approval to allow the system access to the VPN service. The user's approval of the VPN connection request allows the system to automatically scan all apps on the device and set up the VPN connection in the device manager activity class. The denial of a VPN connection by the user automatically stops the setting up of the VPN services. Hence, the system will not monitor the activities of the apps that reside on the device.

The user navigates between the three major activity tabs in the activity diagram depending on the user's selection. If the user selects the app to evaluate activity, the system automatically gets the list of all permissions and intent demanded by each app and executes two different activities simultaneously. The result from the two activities is used to determine the risk category of each app and the risk value associated with each of the apps that reside on the device.

The selection of the detection engine gives the user the ability to navigate between three sub-activities associated with the detection engine. The default sub-activities related to the detection engine are the app's online activities. These sub activities work in conjunction with the device manager VPN services to retrieve all associated activities associated with each app that resides on the device. On the other hand, during the selection of the malicious activities, the system executes activities related to evaluating each app's traffic data and determining malicious connections. In addition, once a malicious activity is detected, the activity is blacklisted. This displays the list of blacklisted traffic data on the user's selection of the blacklisted activities. This selection allows the user to either activate the blacklisted traffic data or not.

D. The Component Diagram of the Prototype System: The component diagram is structural type of UML diagrams that is used to model the physical aspect of the prototype system. The component diagram is like the class diagram although the component diagram focuses on the various components that make up the system by providing a visual specification of each components elements. In this study, the component diagram of the prototype system is presented in Figure 4 in appendix B.

The prototype system comprises three major components: the device manager component, the app evaluator component, and the detection engine component. The device manager component depends on other subsystem components for its basic functionality. This subsystem component includes the VPN connection manager and app information manager components. The app evaluator components rely on a cloud-based ensemble ML model component and a risk score component that resides locally on the device. These subcomponents also depend on other components presented in Figure 4 in appendix B.

On the other hand, the detection engine components have three subsystem components: the online app activities component, the malicious activities component, and the blacklisted activities component. The various subcomponents of the detection engine also use other components to carry out their basic operations. These components include the app network traffic manager, the app network traffic ML model, and the URL global database scanner components.

E. **The Deployment Diagram of the Prototype System:** The deployment diagram is a structural type of UML diagram used to visualize the communication links between the system hardware and software. The deployment diagram shows the execution architecture of the prototype system. The execution architecture includes the various nodes like the hardware or software execution environments and the middleware software connecting the nodes. The deployment diagram of the prototype system is presented in Figure 5 in appendix B. The deployment diagram in Figure 5 in appendix B, shows two execution environment and the device environment. The prototype development execution environment shows the personal computer (PC) hardware where the prototype system was developed. The prototype system is an Android mobile app that depends on two subsystems. The prototype system (MINDPRES) was deployed to a MD with Android 7.0 OS as a minimum software requirement for the device. The two subsystems of the prototype system were developed as an ensemble ML model at the experimental stage of this work. These models were implemented using python programming language and the final models were deployed separately to AWS cloud server.

First, the ensemble ML model uses permission and intent that are declared by the app as features to analyses whether the app is benign or malicious. The ensemble ML model is used

by the app evaluator to determine the riskiness of apps based on the permissions and intent declared by the developers of each apps.

Second, the ensemble ML model that uses the app traffic data (API calls) together with the permission and intents demanded at run-time by each app that resides on the device. This model examines the network traffic data for malicious behaviour of an app activity. Also, the model also extracts URL calls from the traffic data and check the URL against a global malicious URL database and the result from both approach is used to predict whether a specific app traffic is malicious or not. and these models can be used as a web service from the device end.

At the device level, the prototype system communicates with the AWS cloud server via a Transmission Control Protocol (TCP) or Internet Protocol (IP) to consume the web services of the two subsystem of the prototype system that are hosted in the cloud server as presented in the deployment diagram in Figure 5 in appendix B.

5.2.4 THE ALGORITHMIC DESIGN OF THE PROTOTYPE SYSTEM

The general algorithmic description of the three main components of the prototype system are presented in this section. The algorithms presented for each component represent an abstract high-level description of the steps involves in the operation of each major component in the prototype system.

A. Algorithmic Description of the Device Manager

Step 1*: Let **VPNStatus =0** and Initialize VPN Service Connection and Prompt User for permission to monitor all apps*
Step 2: *IF VPN Service Connection is granted by the user, Then Go to Step 3 Otherwise Go to Step*
Step 3: Setup VPN Service Connection for the Device and *Set **VPNStatus**=1 (Ready)*
Step 4: *Scanned all Apps that are Installed by the User and Set Array **DefaultAppList** to AllUserInstallApps*
Step 5: Let ***AppsCount*** = number of records in array ***DefaultAppList and** Set **TotalUserInstallApps = AppsCount***
Step 6*: Stop VPN Service Connection and Set **VPNStatus**=**0***
Step 7*: End of IF structure in Step 2*
Step 8: **OUTPUT**: ***VPNStatus, TotalUserInstallApps***
Step 9: *Exit*

B. Algorithmic Description of the App Evaluator

INPUT: ***DangerousPermissionList***, ***EnsemblePermissionIntentList, PermissionRiskValue***
Step 1*: For Each App X in **DefaultAppList** repeat step 2, 3, 4,5, and 6*
Step 2: Extract the Permission and Intent demanded by app X contained in the Features Listed in ***EnsemblePermissionIntentList*** as ***appPermissionIntentLis**t* and Set array ***PI*** = ***appPermissionIntentLis**t***

Step 3: Extract the dangerous Permission demanded by app X contained in the Features Listed in ***DangerousPermissionList*** as ***appDangerousPermissionLis****t* and Set array **DP** = ***appDangerousPermissionLis****t*
Step 4: *Compute the* ***RiskScore*** *of app X with the selected dangerous permission in* ***DP*** *and Extract the Permission Risk Value for each permission contained in* ***PermissionRiskValue*** *that exists in* ***DP***
Step 5: Get the result of the ensemble ML prediction for app X using the features extracted from Step 2 in array ***PI*** and store the result return in a variable ***MLResult*** as integer (0 is benign and 1 is malicious)
Step 6*: IF* ***RiskScore*** *in Step 4 is greater than or equal to 0.75 and* ***MLResult=0*** *Then Set* ***RiskCategory*** *to* ***"High Risk App"*** *and go to Step 12* ***Otherwise*** *go to step 7*
Step 7*: IF* ***RiskScore*** *in Step 4 is greater than or equal to 0.50 and* ***MLResult=0*** *Then Set* ***RiskCategory*** *to* ***"Medium Risk App"*** *and go to Step 12* ***Otherwise*** *go to step 8*
Step 8*: IF* ***RiskScore*** *in Step 4 is greater than or equal to 0.00 and* ***MLResult=0*** *Then Set* ***RiskCategory*** *to* ***"Low Risk App"*** *and go to Step 12* ***Otherwise*** *go to step 9*
Step 9*: IF* ***RiskScore*** *in Step 4 is greater than or equal to 0.65 and* ***MLResult=1*** *Then Set* ***RiskCategory*** *to* ***"High Risk App"*** *and go to Step 12* ***Otherwise*** *go to step 10*
Step 10*: IF* ***RiskScore*** *in Step 4 is greater than or equal to 0.25 and* ***MLResult=1*** *Then Set* ***RiskCategory*** *to* ***"Medium Risk App"*** *and go to Step 12* ***Otherwise*** *go to step 11*
Step 11*: Set* ***RiskCategory*** *to* ***"Low Risk App"*** *and go to Step 12* ***End of If Structure in Step 6***
Step 12: **OUTPUT** ***RiskScore, RiskCategory***
Step 13: *End of Step 1 For -Loop*
Step 14: *Exit*

C. Algorithmic Description of the Detection Engine

Var: Array<string>: **AppsActivities, MaliciousAppsTraffic, BlackListedAppsTraffic**
INPUT: ***VPNStatus****,* ***EnsemblePermissionIntentList, TrafficDataList, DefaultAppTrafficList***
Step 1: IF **VPNStatus** =1 Then Go to Step 2 Otherwise Go to Step 16
Step 2*: For Each App X in* ***DefaultAppList*** *repeat step 3, 13 and 14*
Step 3*: For Each Traffic Data of App X in* ***DefaultAppTrafficList*** *repeat step 4,5, 6,7,8,9, and 10*
Step 4: Extract the Permission and Intent demanded by app X at run-time whenever an online Request is made contained in the Features Listed in ***EnsemblePermissionIntentList*** as ***appPermissionIntentLis****t* and Set array ***PI*** = ***appPermissionIntentLis****t*
Step 5: Extract the Traffic Network Data by app X contained in the Features Listed in ***TrafficDataList*** as ***appTrafficData*** and Set array **TD** = ***appTrafficData*** *and add Traffic data for app x in array* **AppsActivities**
Step 6: Extract the URL call by app X contained in the ***TrafficDataList*** as ***appTrafficURL*** and Set array **TURL** = ***appTrafficURL***
Step 7: Construct App network traffic dataset from **TD** in step 5 and **PI** in step 4 and set **AppTrafficMLData** as the new dataset for each traffic request consisting of API calls, Permissions, and Intent
Step 8: Get the result of the ensemble ML prediction for app X using the features constructed from Step 7 in array **AppTrafficMLData** and store the result return in a variable ***MLTrafficResult*** as integer (0 is benign and 1 is malicious)
Step 9: Get the result of the Malicious Global Database scanner for app X using the URL extracted from Step 6 in array **TURL** and store the result return in a variable ***URLTrafficResult*** as integer (0 is benign and 1 is malicious)
Step 10*: IF* ***MLTrafficResult*** *=1 OR* ***URLTrafficResult =1*** *Then* ***ADD*** *the traffic data in* **TD** *from step 5 for app X to* **MaliciousAppsTraffic** *and also automatically block* ***TD*** *and* ***ADD*** *to* **BlackListedAppsTraffic** for app X and *go to Step 11* ***Otherwise*** *go to step 12*

Step 11: ***End of If Structure in Step 10***
Step 12: End of Inner For-Loop in Step 3
Step 13: Set **TotalAppActivites** = Total Number of Records in **AppsActivities** in Step 5 Set **TotalMaliciousAppActivites**= Total Number of Records in **MaliciousAppsTraffic** in Step 10 Set **TotalBlacklistedAppActivites**= Total Number of Records in **BlackListedAppsTraffic** in Step 10
Step 14: **OUTPUT: TotalAppActivites, TotalMaliciousAppActivites, TotalBlacklistedAppActivites, AppsActivities, MaliciousAppsTraffic, BlackListedAppsTraffic** for app X.
Step 15: *End of Outer For -Loop **in** Step 2*
Step 16: *Exit*

5.3 PROTOTYPE TESTING

The testing of the prototype system follows a scenario-based approach that demonstrates the functionality of the prototype system implemented as a proof of concept. In this regard, five dummy apps were designed to test how the system would behave in each scenario. The first four scenarios focus on the app evaluator module. This module enables MCC users to assess the risk of Android apps that reside on their device and provides information that helps the user identify each app's risk category and score. At the same time, the last scenarios focus on the detection engine and how the system can automatically detect malicious activities of an app both when the device is active and when the device is idle.

The app evaluator in the prototype system (MINDPRES) analyses all user-installed apps that reside on an Android device and determines an app's risk score and category (i.e., **High, Medium, or Low)**. The risk category of an app is determined by combining the result of the ensemble ML model classification with the permissions and intents demanded by apps, as shown in Table 5.6, to determine if the app is malicious or benign. The risk score of user-installed apps is determined by a probability value function of the statistical model discussed in equations Eq. 5.1 to Eq. 5.8 using only the dangerous permissions. The statistical model uses equations Eq. 5.7 and Eq. 5.8 to determine the risk category of each app, as presented in Table 5.8.

Table 5.6 Selected Permissions and Intent used in the design of the ensemble ML model

ID	Permission and Intent Name	ID	Permission and Intent Name
PI1	WRITE_EXTERNAL_STORAGE	PI21	CHANGE_WIFI_STATE
PI2	READ_PHONE_STATE	PI22	DISABLE_KEYGUARD
PI3	ACCESS_COARSE_LOCATION	PI23	KILL_BACKGROUND_PROCESSES
PI4	ACCESS_FINE_LOCATION	PI24	MODIFY_AUDIO_SETTINGS
PI5	GET_TASKS	PI25	READ_CONTACTS
PI6	READ_EXTERNAL_STORAGE	PI26	READ_SMS
PI7	SYSTEM_ALERT_WINDOW	PI27	RECEIVE_BOOT_COMPLETED
PI8	READ_LOGS	PI28	RESTART_PACKAGES
PI9	MOUNT_UNMOUNT_FILESYSTEMS	PI29	VIBRATE
PI10	CAMERA	PI30	WAKE_LOCK
PI11	RECORD_AUDIO	PI31	BOOT_COMPLETED
PI12	GET_ACCOUNTS	PI32	PACKAGE_REMOVED
PI13	CALL_PHONE	PI33	SEARCH
PI14	WRITE_SETTINGS	PI34	USER_PRESENT
PI15	SEND_SMS	PI35	VIEW
PI16	ACCESS_LOCATION_EXTRA_COMMAND	PI36	BROWSABLE
PI17	ACCESS_WIFI_STATE	PI37	DEFAULT
PI18	BROADCAST_STICKY	PI38	HOME
PI19	CHANGE_CONFIGURATION	PI39	INFO
PI20	CHANGE_NETWORK_STATE		

Table 5.7. Risk Score Output (Dangerous Permissions) of the Statistical Model

ID	Risk Score
P1	0.5769
P2	0.8460
P3	0.7154
P4	0.4982
P5	0.6063
P6	0.4816
P7	0.2557
P8	0.4991
P9	0.5091
P10	0.3986
P11	0.1846
P12	0.3375
P13	0.2115
P14	0.2635
P15	0.3707

Table 5.8 Classification Outcome of the App Evaluator

ML Prediction	Risk Score of an App	Final Risk Category Classification
Benign App	$RiskScore \in (0.00,0.65)$	Low Risk App
Benign App	$RiskScore \in (0.66,0.85)$	Medium Risk App
Benign App	$RiskScore \in (0.86,1.00)$	High Risk App
Malicious App	$RiskScore \in (0.00,0.25)$	Low Risk App
Malicious App	$RiskScore \in (0.26,0.60)$	Medium Risk App
Malicious App	$RiskScore \in (0.61,1.00)$	High Risk App

5.3.1 TEST CASE SCENARIO 1

The test case scenario 1 illustrates an app with a zero-risk score classified as low risk by the prototype system. The app designed for scenario 1 did not use any dangerous permissions. For example, suppose an Android app is designed for a student course registration system at a university. The app developers demanded the permissions and intents contained in Table 5.9 for the first version of the software release to be used by the students.

The second column in Table 5.9 shows the name of the permissions or intents demanded by the app. The third column shows the equivalent permission or intent ID as part of the selected features used in the design of the ensemble ML model as shown in Table 5.6, and the fourth column shows the equivalent dangerous permission risk score used in the design of the risk score computation of each app as shown in Table 5.7.

Table 5.9 Permission Requested by the Test App for Scenario 1

S/N	Permission/ Intent Name	ML Features	Risk Score
1	INTERNET	-	-
2	VIBRATE	PI29	-
3	ACCESS_NETWORK_STATE	-	-
4	ACCESS_WIFI_STATE	PI17	-
5	WAKE_LOCK	PI30	-
6	ACTION_MAIN	-	-
7	CATEGORY LAUNCHER	-	-

Note - The permission or intent is not part of the selected features used for design of the ensemble ML model or the statistical model.

For evaluation purpose, the test case app is an empty app that does nothing but simulate the permission and intent demanded by such app in this scenario

Outcome of the Test in Scenario 1

A. Ensemble ML model returns the value of zero which means the app is classified as Benign

B. The statistical model returns a risk score of zero (0.00) for the app in this scenario. Since none of the permissions demanded by the app in scenario 1 belongs to the dangerous permissions listed in table 5.7.

C. The app evaluator classified the test app in scenario 1 as a low-risk app with 0.00 risk score.

5.3.2 TEST CASE SCENARIO 2

The test case in scenario 2 illustrates the modification of the app in scenario 1 to handle some extra functionalities. This modification changes the risk score of the app. For example, suppose the Android app in Scenario 1 is modified by the developer to include some additional functionalities. The updated version of the app demanded the following permissions and Intent as contained in Table 5.10 for the second version of the software release for used by the student.

The second column in Table 5.10 shows the name of the permission or intent demanded by the app. The third column shows the equivalent permission or intent ID as part of the selected features used in the design of the ensemble ML model as shown in Table 5.6 and the fourth column shows the equivalent dangerous permission risk score used in the design of the risk score computation of each app as shown in Table 5.7

Table 5.10 Permission Requested by the Test App for Scenario 2

S/N	Permission/ Intent Name	ML Features	Risk Score
1	INTERNET	-	-
2	VIBRATE	PI29	-
3	ACCESS_NETWORK_STATE	-	-
4	ACCESS_WIFI_STATE	PI17	-
5	PERMISSION WAKE_LOCK	PI30	-
6	ACTION_MAIN	-	-
7	CATEGORY LAUNCHER	-	-
8	READ_EXTERNAL_STORAGE	PI6	0.4816
9	CAMERA	PI10	0.3986

Note - The permission or intent is not part of the selected features used for design of the ensemble ML model or the statistical model.

For evaluation purpose, the test app is an empty app that does nothing but demonstrates the permissions and intents demanded by such app in this scenario

Outcome of the Test in Scenario 2

A. Ensemble ML model returns zero which means the app is classified as Benign

B. The statistical model returns a risk score of 0.44 because of the average risk value of the dangerous permissions demanded by the app, as shown in table 5.7. The dangerous camera permission does not pose much more threat to the device user than the read external storage permission. The app can only read information stored on the internal memory, such as images combined with the camera permission that allows the app to take a photograph of an object.

C. The app evaluator classified the test app in scenario 2 as low-risk app with the risk score of 0.44.

5.3.3 TEST CASE SCENARIO 3

The test case in scenario 3 illustrates the modification of the app in scenario 1 to handle some extra functionalities. This modification changes the risk score of the app. For example, suppose the Android app in Scenario 1 is modified by the developer to include some additional functionalities. The updated version of the app demanded the following permissions and Intent as contained in Table 5.11 for the third version of the software release for used by the student.

The second column in Table 5.11 shows the name of the permission or intent demanded by the app. The third column shows the equivalent permission or intent ID as part of the selected features used in the design of the ensemble ML model as shown in Table 5.6 and the fourth column shows the equivalent dangerous permission risk score used in the design of the risk score computation of each app as shown in Table 5.7.

Table 5.11 Permission Requested by the Test App for Scenario 3

S/N	Permission/ Intent Name	ML Features	Risk Score
1	INTERNET	-	-
2	VIBRATE	PI29	-
3	ACCESS_NETWORK_STATE	-	-
4	ACCESS_WIFI_STATE	PI17	-
5	WAKE_LOCK	PI30	-
6	IACTION_MAIN	-	-
7	CATEGORY LAUNCHER	-	-
8	READ_PHONE_STATE	PI2	0.8460
9	ACCESS_FINE_LOCATION	PI4	0.4982
10	BROWSABLE	PI36	-

Note - The permission or intent is not part of the selected features used for design of the ensemble ML model or the statistical model.

For evaluation purpose, the test app is an empty app that does nothing but demonstrate the permissions and intents demanded by such app in this scenario

Outcome of the Test in Scenario 3

A. Ensemble ML model returns zero which means the app is classified as Benign

B. The statistical model returns a risk score of 0.67 because of the average risk value of the dangerous permissions demanded by the app, as shown in table 5.7. The increase in the risk score of the app is because of the risk value associated with the two dangerous permissions. For example, the Read Phone State allows the apps' developers to access sensitive information on the device. Such as the current cellular network information, the status of any ongoing calls, and a list of any phone accounts registered on the device. The Access Fine Location permission allows the developers of the app to access an exact location of the device at any point in time when the user grant the app access to such permission on the device.

C. The app evaluator classified the test app in scenario 3 as medium-risk app with the risk score of 0.67.

5.3.4 TEST CASE SCENARIO 4

The test case in scenario 4 illustrates the modification of the app in scenario 1 to handle some extra functionalities. This modification changes the risk score of the app. For example, suppose the Android app in Scenario 1 is modified by the developer to include some additional functionalities. The updated version of the app demanded the following permissions and Intent as contained in Table 5.12 for the fourth version of the software release for used by the student.

The second column in Table 5.12 shows the name of the permission or intent demanded by the app. The third column shows the equivalent permission or intent ID as part of the selected features used in the design of the ensemble ML model as shown in Table 5.6 and the fourth column shows the equivalent dangerous permission risk score used in the design of the risk score computation of each app as shown in Table 5.7

Table 5.12 Permission Requested by the Test App for Scenario 4

S/N	Permission/ Intent Name	ML Features	Risk Score
1	PERMISSION INTERNET	-	-
2	PERMISSION VIBRATE	PI29	-
3	PERMISSION ACCESS_NETWORK_STATE	-	-
4	PERMISSION ACCESS_WIFI_STATE	PI17	-
5	PERMISSION WAKE_LOCK	PI30	-
6	INTENT MAIN	-	-
7	INTENT CATEGORY LAUNCHER	-	-
8	PERMISSION READ_PHONE_STATE	PI2	0.8460
9	PERMISSION ACCESS_FINE_LOCATION	PI4	0.4982
10	INTENT BROWSABLE	PI36	-
11	PERMISSION GET_TASKS	PI5	0.6063
12	PERMISSION WRITE_EXTERNAL_STORAGE	PI1	0.5769

Note - The permission or intent is not part of the selected features used for design of the ensemble ML model or the statistical model.

For evaluation purpose, the test app is an empty app that does nothing but demonstrates the permission and intent demanded by such app in this scenario

Outcome of the Test in Scenario 4

A. The Ensemble ML model returns one, which means the app is classified as malicious. The result of the ensemble ML model is evidence of the app using dangerous permissions commonly

used by malicious apps to achieve some form of control over sensitive information stored on the device. The introduction of Get_Task permissions allows apps to read the current tasks or lists of all other apps and their current tasks on the device. Combining such permission by the app with the ability to write and read from external storage allows the app to manipulate information stored on the device. The access to the internet permission also means the app can send information out of the device without the user's knowledge. Also, the Read Phone State allows the developers to access sensitive information on the device. Such as the current cellular network information, the status of any ongoing calls, and a list of any phone accounts registered on the device. The Access Fine Location permission allows the developers of the app to access the device's exact location at any point in time when the user grants the app access to such permission on the device. These four dangerous permissions, commonly used by malicious apps, give the developer great control of the device if all these permissions are granted without the user knowing the developer's intention.

B. The statistical model returns a risk score of 0.63 because of the average risk value of the dangerous permissions demanded by the app, as shown in table 5.7. The statistical model's decrease in the risk score value of the app return is because of its average, although this does not have any significant effect on the classification. The result of the ensemble ML model has shown that the more dangerous permission requested by an app, the more likely the app is to be classified as malicious. Therefore, in this study, the risk score of an app greater than 0.65 is classified as a medium-risk or high-risk app, depending on the outcome of the ensemble ML model.

C. The app evaluator classified the test app in scenario 4 as a "high-risk app" with a 0.63 risk score. Although Scenario 3 has a higher risk score value and is classified as a medium-risk app because the ML model predicted it as a benign app. However, scenario 4 is different, with a lower risk score of 0.63 compared to 0.67 in scenario 3. In this case, the risk category of the app described in Scenario 4 is high-risk because the ML model classifies the app as malicious. Hence, the resultant outcome of the app evaluator for this scenario is a high-risk app.

5.3.5 TEST CASE SCENARIO 5

In scenario 5, the detection engine uses the behaviour patterns of the traffic data generated by an app whenever a request is made to a specific domain to monitor abnormalities in the device's behaviour. The detection engine used an ensemble ML model that required the permission and intent demanded by an app at run-time together with network traffic data to determine if the request was malicious or not. The detection engine also used a global blacklisted malicious API to screen the domain each network traffic request was calling from the device and used both results to determine if the call was malicious or not. This scenario demonstrates how an app might behave when called with a malicious URL.

For example, suppose the updated version of the app in Scenario 4 is loaded with malicious code unknown to the user. Using the same set of permissions in table 5.12 with an additional intent

USER_PRESENT enables the app to check if the user is using the device or not. The behaviour of the app when the user is using the device is different from when the user is not using the device as described below:

Phase 1: When the user is using the device, the app in Scenario 5 reads the user's location information and sends it to the remote server (maybe the user's email address) accessible by the user, which is a legitimate URL.

Phase 2: When the user is not using the device, the app in scenario 5 reads the user location information with the list of all apps installed on the user device and send it to the remote server accessible by the developer, a legitimate URL. Suppose the remote server with the URL in that domain has been reported and listed in the global database of malicious URL.

Outcome of the Test in Scenario 5

A. Phase 1. MINDPRES will flags all request as benign since the URL is benign.
B. Phase 2. MINDPRES will flags all request as malicious since the URL is malicious as contained in global database of malicious URL and automatically block all calls to that URL from the device.

5.4 CHAPTER SUMMARY

This chapter describes the implementation details of the prototype system and the various sub-systems design. The database design of the prototype system UIs was presented. The tools used in developing the prototype system are discussed in this chapter. The UML diagrams, which show the various system objects and how they interact, were discussed in this chapter. In addition, a detailed algorithmic design of the main components of the prototype system was also presented. The final parts of this chapter discuss five different test case scenarios used to demonstrates the functionalities of the implemented prototype system and how the various sub modules work together. The test case scenarios use a dummy app to demonstrates how changes in permissions and intents requested by an app can affect the risk scores and categories when screened by the implemented prototype system (MINDPRES).

CHAPTER SIX

PROTOTYPE EVALUATION

The previous chapter discusses the design and implementation of the prototype system. The software tools used for implementing the prototype were discussed. In addition, the UML diagrams showing the various component implementations of the prototype system were discussed. The previous chapter also discusses the various algorithms used to implement the different modules that make up the prototype system. The description of the different test case scenarios, that shows how the prototype system was tested and the outcome of each case scenarios were discussed.

This chapter describes how the prototype system (MINDPRES) implemented as a proof of concept was evaluated. The prototype is evaluated using real-life Android MDs, and the results obtained from the experiment were evaluated using a confusion matrix. The MDs' energy consumption was recorded during the experiment, and the results obtained were evaluated. The prototype system is evaluated by invited IT security experts, whose expert opinions are also presented in this chapter.

6.1 PROTOTYPE EVALUATION

The prototype system was evaluated based on its performance and energy consumption requirements. First, the prototype system's performance evaluation (MINDPRES) uses a hybrid analysis approach (both static and dynamic analysis). Second, the energy consumption of the prototype system was evaluated to assess the app energy consumption rate and its feasibility to cope with resource-constrained MDs in the MCC environment.

To evaluate the performance of the prototype system, several real-life experiments were conducted. The performance evaluation of the prototype system follows two phases.

The first phase involves the risk assessment of apps that reside on the device by the app evaluator to determine the risk category of the various apps that reside on user device.

The second phase involves monitoring the actual network behaviours of all apps that reside on the device by listening to their network activities and using an ensemble ML model trained with network data to detect and prevent malicious activities. To evaluate the performance of the detection engine in the prototype system, the validation metrics of the confusion matrix are discussed in Chapter Four, Sections 4.3.2 and 4.3.3. equations Eq 4.1 to Eq 4.8 were used to validate the 1,000 mobile apps installed on five Android devices.

6.1.1 Experimental Setup For The Prototype Performance Evaluation

This experiment aims to evaluate the performance of the prototype system (MINDPRES) using the evaluation metrics stated in Chapter Four, Section 4.3.2. The experimental setup for evaluating the prototype system uses 1000 (600 benign and 400 malicious) apps as a testbed. The distribution of the app samples installed on the five devices used for this experiment is shown in Table 6.1.

The evaluation of the prototype system (MINDPRES) uses five Android devices with the following hardware configuration.

A. ***Device A***: EE Tablet HTC Nexus 9.8.9, 1.8GB RAM 32GB Internal storage

B. ***Device B***: Samsung Galaxy Tab A (SM-T380) 2GB RAM, 16GB Internal storage

C. ***Device C***: Samsung Galaxy Tab A (SM-T380) 2GB RAM, 16GB Internal storage

D. ***Device D:*** Samsung Galaxy Tab A (SM-T380) 2GB RAM, 16GB Internal storage

E. ***Device E***: Samsung Galaxy Tab A (SM-T380) 2GB RAM, 16GB Internal storage

Table 6.1 Apps Distribution Sample Installed in Each Device

DEVICE	Benign Apps (Google Play Store)	Malicious Apps (CICMalDroid2020)
A	240	200
B	90	50
C	90	50
D	90	50
E	90	50
Total	**600**	**400**

6.1.2 Description Of The Evaluation Testbed

The evaluation of the prototype system uses a testbed dataset that is different from the initial training and testing datasets used in the laboratory experiments to evaluate the performance of the ensemble ML model discussed in Chapter Four. The 600 benign samples were obtained from popular apps available in the Google Play store with at least one million users' downloads. The apps were downloaded and installed on each device between November 1st, 2021, and December 1st, 2021. The evaluation of the system uses the top thirty most popular apps downloaded from each of the twenty selected app categories in the Google Play Store. These apps categories include most apps that are found in almost every user device that are used for their day-to-day activities such as social networking, business, travel, education e.t.c.

The testbed used for the evaluation of the prototype system contains 400 malicious app samples. These samples were obtained from a recent malware repository (CICMalDroid2020) publicly available to evaluate the malware detection system (https://www.unb.ca/cic/datasets/maldroid-2020.html). The malicious APK samples in the CICMalDroid2020 dataset contains four malware categories which are briefly describes as follows:

A. ***Adware***: The adware category in this dataset includes advertisement materials (i.e., ads) that hide behind legitimate apps. The ad libraries contain malware that repeatedly runs on the device. Even though the user tries to close the ads, they keep popping up. It can infect the device and get root-level access, compromising the user device and gaining unauthorized access to sensitive data or device operation (Mahdavifar et al., 2020).

B. ***Banking Malware***: The banking malware, this malware app category is trojan-based designed to mimic the original banking app interface to infiltrate user devices, gain unauthorized access to use banking apps, and steal sensitive information from the device.

C. ***SMS Malware***: The SMS malware category; this app is designed to intercept payload operations to conduct attacks on the device. These malware categories send malicious SMS to users, intercept the SMS and steal data from the device.

D. ***Riskware***: The riskware malware category includes legitimate apps manipulated by users to take the form of mobile malware, SMS malware, or adware malware. The malicious app category can install new apps on the device unknown to the user.

The benign and malicious apps used for the evaluation were checked using the VirusTotal services to ascertain whether they were malicious or not. The malware samples use a minimum of 15 antiviruses as benchmark criteria for selecting the app as malicious (i.e., at least 15 antivirus engines in VirusTotal Services must flag the app apk as malicious). The benign samples obtained from the Google Play store were also checked using the same VirusTotal services, with none of the antivirus flagging the app as malicious. After checking both benign and malicious apps using the VirusTotal service, each app was installed on the five devices, as shown in Table 6.1. The distribution of the benign and malicious apps' samples and the numbers of apps installed on each device are shown in Tables 6.2 and 6.3.

Table 6.2 Benign Apps Downloaded from Google Play Store

ID	Category	Total Apps	Average Downloads	Average Ratings	Device
B1	Watch	30	95 million	4.3	
B2	Art and Design	30	21 million	4.3	
B3	Beauty	30	1 million	4.4	**A**
B4	Business	30	100 million	4.3	
B5	Communication	30	200 million	4.1	
B6	Education	30	20 million	4.4	
B7	Events	30	1 million	3.8	
B8	Food and Drink	30	10 million	4.2	
B9	Shopping	30	50 million	4.5	**B**
B10	Social	30	500 million	4.0	
B11	News & Magazines	30	5 million	4.2	
B12	Finance	30	10 million	3.8	**C**
B13	Entertainment	30	100 million	4.1	
B14	Lifestyle	30	10 million	4.2	
B15	Music & Audio	30	50 million	4.4	**D**
B16	Maps & Navigation	30	10 million	4.2	
B17	Travel and Local	30	1 million	4.4	
B18	Tools	30	10 million	4.3	**E**
B19	Sports	30	10 million	4.2	
B20	Dating	30	10 million	3.9	

Table 6.3 Malicious Apps Downloaded from CICMalDroid2020

ID	Category	Total Apps	Device
M1	Adware	100	A
M2	Banking Malware	100	A
M3	SMS Malware	50	B
M4	SMS Malware	50	C
M5	Mobile Riskware	50	D
M6	Mobile Riskware	50	E

6.1.3 PHASE ONE (THE APP EVALUATOR) RESULTS

The app evaluator assesses the risk of each user-installed app on the device in the first phase of the prototype evaluation. The risk assessment experiment was conducted, following the complete installation of all apps on the various devices. The prototype system (MINDPRES) was installed on the five devices (A, B, C, D, and E). The app evaluator evaluated the risk category and the ensemble ML classification result using the static analysis approach of all apps on the device. The results obtained during the evaluation process are recorded and presented in Table 6.4.

Table 6.4 Prototype System Risk Assessment Evaluation Result using Both Benign and Malicious Apps

Category	MLCR-Benign	MLPR-Malicious	Low Risk	Medium Risk	High Risk	Total Apps
B1	30	0	29	1	0	30
B2	30	0	30	0	0	30
B3	30	0	30	0	0	30
B4	28	2	28	2	0	30
B5	27	3	27	3	0	30
B6	28	2	28	1	1	30
B7	30	0	30	0	0	30
B8	30	0	30	0	0	30
B9	26	4	26	3	1	30
B10	25	5	25	5	0	30
B11	28	2	26	4	0	30
B12	29	1	29	1	0	30
B13	29	1	29	1	0	30
B14	29	1	28	2	0	30
B15	27	3	27	3	0	30
B16	28	2	28	2	0	30
B17	29	1	29	1	0	30
B18	28	2	28	2	0	30
B19	28	2	27	3	0	30
B20	29	1	29	1	0	30
M1	7	93	0	73	27	100
M2	3	97	0	85	15	100
M3	4	46	2	35	17	50
M4	2	48	0	30	16	50
M5	3	47	0	38	16	50
M6	1	49	1	33	12	50

Note: MLCR: Machine Learning Classification Result

The results presented in Table 6.4 show that all selected apps in categories B1, B2, B3, B7, and B8 were correctly classified as benign by the ensemble ML model in the prototype system using the static analysis approach. These results show an improved performance by accurately evaluating apps in these categories by the prototype system. At the same time, the results presented in Table 6.4 show that the ensemble ML model embedded in the app evaluator can effectively distinguish benign apps from malicious apps with over 90% classification accuracy at all times. Overall, out of 600 benign apps downloaded from the Google Play store, 568 were correctly classified as benign apps, with 32 wrongly classified as malicious apps. Furthermore, the benign apps that were wrongly classified as malicious were found to demand dangerous permissions and intents commonly used by malicious apps. The developers of these apps can misuse the permissions requested by these apps for malicious purposes if their intentions change in the future. The end user must be careful with the granting of dangerous permissions whenever an app requested it, and they also need to trust the source before granting such permission to prevent the compromization of their device.

In addition, the ensemble ML classification results for the malicious app set used in the evaluation in the results shown in Table 6.4 show that 380 (95%) apps from a total of 400 apps from malicious

app categories M1, M2, M3, M4, M5 and M6 were correctly classified as malicious apps. In contrast, only 20 apps from these categories were wrongly classified as benign. A further investigation into this set of malicious apps shows that these apps did not use dangerous permissions commonly used by malicious apps. This set of apps used dangerous permissions and intents commonly used by benign apps. These apps can evade detection if a system only uses a static analysis approach to distinguish between malicious and benign apps without monitoring their actual behaviour at run time. However, the prototype system implemented in this study can detect such apps using the hybrid approach that combines both the static and dynamic analysis approach. The dynamic analysis approach of the prototype system implemented in this study can monitor apps actual behaviour and it can easily detect apps that falls under these categories.

The risk assessment results of the benign apps used in the prototype evaluation is shown in Figure 6.1. The results show that all apps in categories B2, B3, B7, and B8 are low risk, with risk scores between 0.00 and 0.65. The raw data from the results of the benign apps used in the evaluation shows that about 5% of all benign apps were classified as zero risk (risk score 0.00) and other apps had a risk score of greater than 0.00 but less than or equal to 0.65. Overall, 563 out of the 600 of the benign apps' samples used in the evaluation were classified as low risk, 35 as medium risk, and only 2 benign apps from categories B6 and B9 were classified as high-risk apps with a risk score of above 0.60. This is because these 2 benign apps were wrongly classified as malicious by the ensemble ML model using the static analysis approach. The results of the app evaluators have shown that 94% of the benign app samples used for this evaluation are classified as low-risk apps.

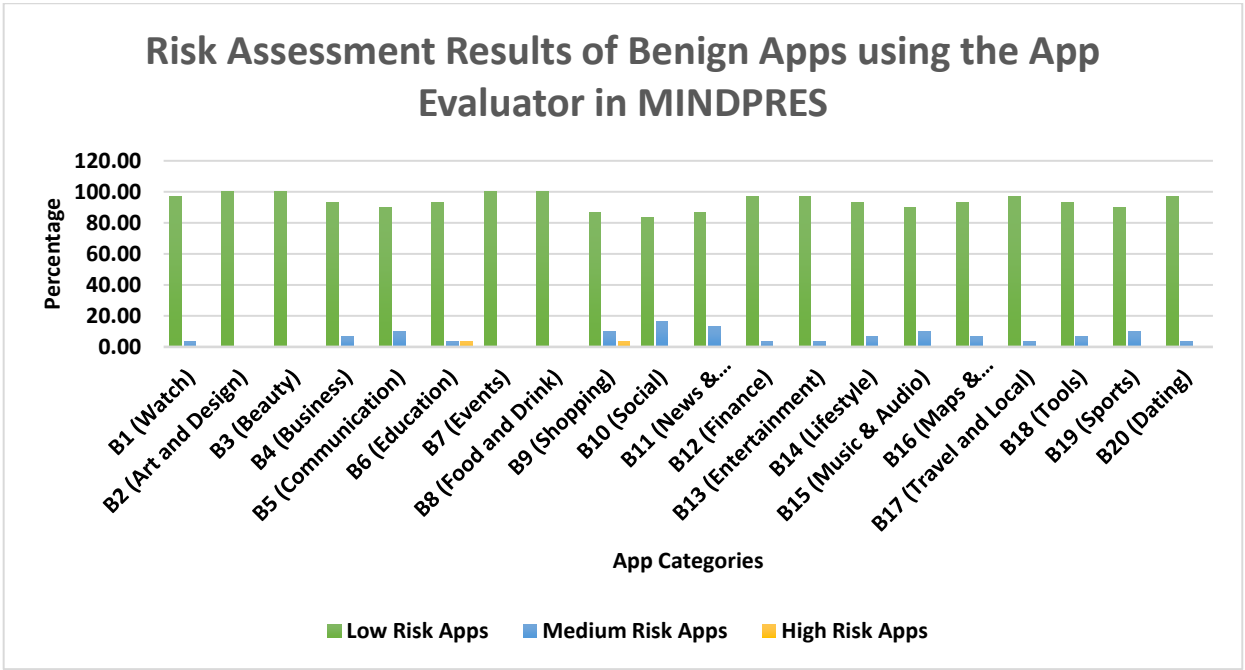

Figure 6.1 Risk Assessment Result of all Benign apps used in the Evaluation

The risk assessment results regarding the risk categorization (high, medium, or low) of the malicious apps used in the evaluation are presented in Figure 6.2. The results presented in Figure 6.2 show that all apps in categories M1, M2, M4 and M5 are classified as either medium-risk or high-risk with a risk score of between 0.25 to 1.00. On the other hand, only 3 apps from categories M3 and M6 were low risk. These results show the prototype system's weakness in using a static analysis approach for malware detection, as these apps were designed to evade detection by such a system. Overall, 294 of the malicious apps' categories were classified as medium risk, 103 as high risk, and only 3 malicious apps were classified as low-risk apps, with a risk score of 0.00. The above performance results of the app evaluators have shown that 99% of the malicious apps samples used for the prototype evaluation are classified as either medium-risk or high-risk apps.

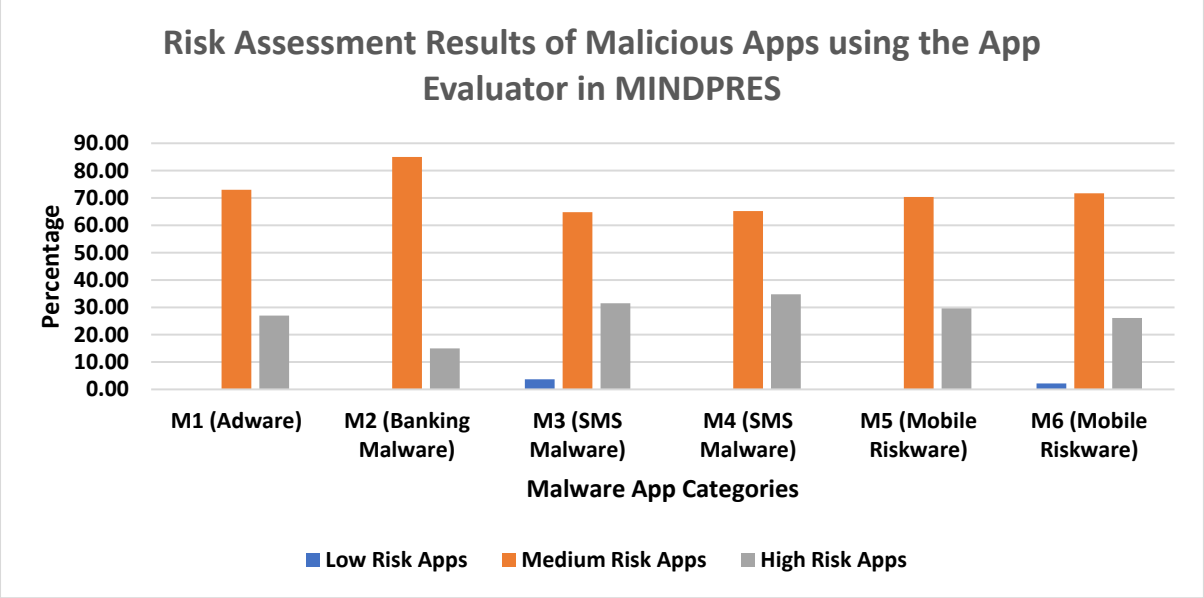

Figure 6.2 Risk Assessment Result of all Malicious apps used in the Evaluation

6.1.4 PHASE TWO (THE DETECTION ENGINE) RESULTS

The second phase of the prototype evaluation focuses on the detection engine subsystem. The detection engine experiments aimed to evaluate the performance of the prototype system using the hybrid analysis approach that combines both the static and dynamic analysis approach. The detection engine performance is based on the actual behaviour of the apps both when the user is using the device and when the device is idle. The experiment was carried out for two weeks to evaluate the detection engine performance of the prototype system (MINDPRES) installed in the device.

During the experiment, each device was used for 2 hours daily. The usage of the device includes the running of the all apps that reside on the device for the prototype system to monitor the actual network behaviours of each app as regards the external API calls and other network-related activities

requested by the app. The prototype system has a VPN service that allows the system to monitor all network activities of all apps on the device. The detection engine logs the details of each network call from each app on the device and stores the results in a local database on the device. The detection engines also analyse each network call made by the various apps for malicious activities using an ensemble ML model trained with network data using the hybrid analysis approach. The detection engine also uses a global database containing known malicious URLs to assess each domain URL request made by the apps on the device. The notification module of the detection engine notifies the user if a malicious activity (such as spamming, phishing, botnet or malware) is detected in any request made by the various apps. The detection of any malicious activities by the detection engine automatically triggers the prevention modules to blocks the detected malicious activities and allows the user to either enable it in the event of false alarms. The details on the number of activities recorded both when the device was in use and when the device was idle in this experiment are shown in Table 6.5.

Table 6.5 Network Activities of all Apps in the Device Captured by the Detection Engine

DEVICE	Actual App Type	Activities When Device is in Used	Activities When Device is Idle	No of Malicious Activities	Total apps with Malicious Activities
A	Benign	1,978	597	13	6
B	Benign	899	215	5	2
C	Benign	768	198	2	1
D	Benign	987	231	7	3
E	Benign	1204	149	4	2
A	Malicious	2876	459	1781	187
B	Malicious	768	202	571	48
C	Malicious	1377	599	650	45
D	Malicious	1422	231	679	49
E	Malicious	1009	456	752	42

6.1.5 PERFORMANCE EVALUATION RESULTS

The overall performance of the prototype system (MINDPRES) in this study is evaluated by the experimental results presented in Sections 6.1.3 and 6.1.4 (Table 6.4 and Table 6.5). The evaluation is based on the prototype system ability to detect intrusion activities caused by malicious apps at the user layer of the MCC environment. The evaluation of the prototype system uses the confusion matrix (TP, FP, TN, and FN) defined in Chapter Four, Section 4.3.2 and employs equations Eq 4.1 to Eq 4.8 to evaluate the performance of the system. The detection performance of the system is evaluated in two phases. The first phase uses the static analysis approach with permissions and intents demanded by apps as features to evaluate the performance of the ensemble ML model in the app evaluator module in the prototype system. The results used for the first phase are presented in Table 6.4, columns 1 to 3. The second phase of the evaluation uses a hybrid analysis approach (both static and dynamic analysis) in the detection engine to assess whether an app activity is

malicious or not by monitoring the actual behaviour and using the signature-based approach to evaluate whether the URL the app calls is contained in a global database of malicious URLs.

The results presented in Table 6.5 are details of the network-related activities and detection engine results of the experiment conducted for two weeks. Using the results in Table 6.4 and Table 6.5, the performance evaluation results of the prototype system applying the confusion matrix and its evaluation metrics defined in Chapter Four equations Eq. 4.1 to 4.8 are shown in Table 6.6. The results are in Table 6.6, which shows the detailed evaluation results of the two approaches used for the experiment. The evaluation result of the prototype system is shown in Figure 6.3. Overall, the prototype system achieved above 90% classification accuracy in all devices using both the static and hybrid approaches. The static approach achieves better classification accuracy than the hybrid in devices A, C, and E. Although in devices B and D, the hybrid approach outperforms the static approach. The detection performance using the precision rate (PR) shows that the hybrid approach performs better in detecting malicious activities than using the static approach, with over 94% on all devices used for the evaluation. Similarly, the hybrid approaches have a better false-positive rate in all devices, as low as 1.11% in device C.

Table 6.6 Detection Performance Evaluation Results of the Prototype System

Device	Approach	TP	FP	TN	FN	CA	ER	PR	RC	FM	FPR	FNR	FAR
A	Static	190	7	233	10	96.14	3.86	96.45	95.00	95.72	2.92	5.00	3.96
	Hybrid	187	6	234	13	95.68	4.32	96.89	93.50	95.17	2.50	6.50	4.50
B	Static	46	4	86	4	94.29	5.71	92.00	92.00	92.00	4.44	8.00	6.22
	Hybrid	48	2	88	2	97.14	2.86	96.00	96.00	96.00	2.22	4.00	3.11
C	Static	48	3	87	2	96.43	3.57	94.12	96.00	95.05	3.33	4.00	3.67
	Hybrid	45	1	89	5	95.71	4.29	97.83	90.00	93.75	1.11	10.00	5.56
D	Static	47	6	84	3	93.57	6.43	88.68	94.00	91.26	6.67	6.00	6.33
	Hybrid	49	3	87	1	97.14	2.86	94.23	98.00	96.08	3.33	2.00	2.67
E	Static	49	5	85	1	95.71	4.29	90.74	98.00	94.23	5.56	2.00	3.78
	Hybrid	42	2	88	8	92.86	7.14	95.45	84.00	89.36	2.22	16.00	9.11

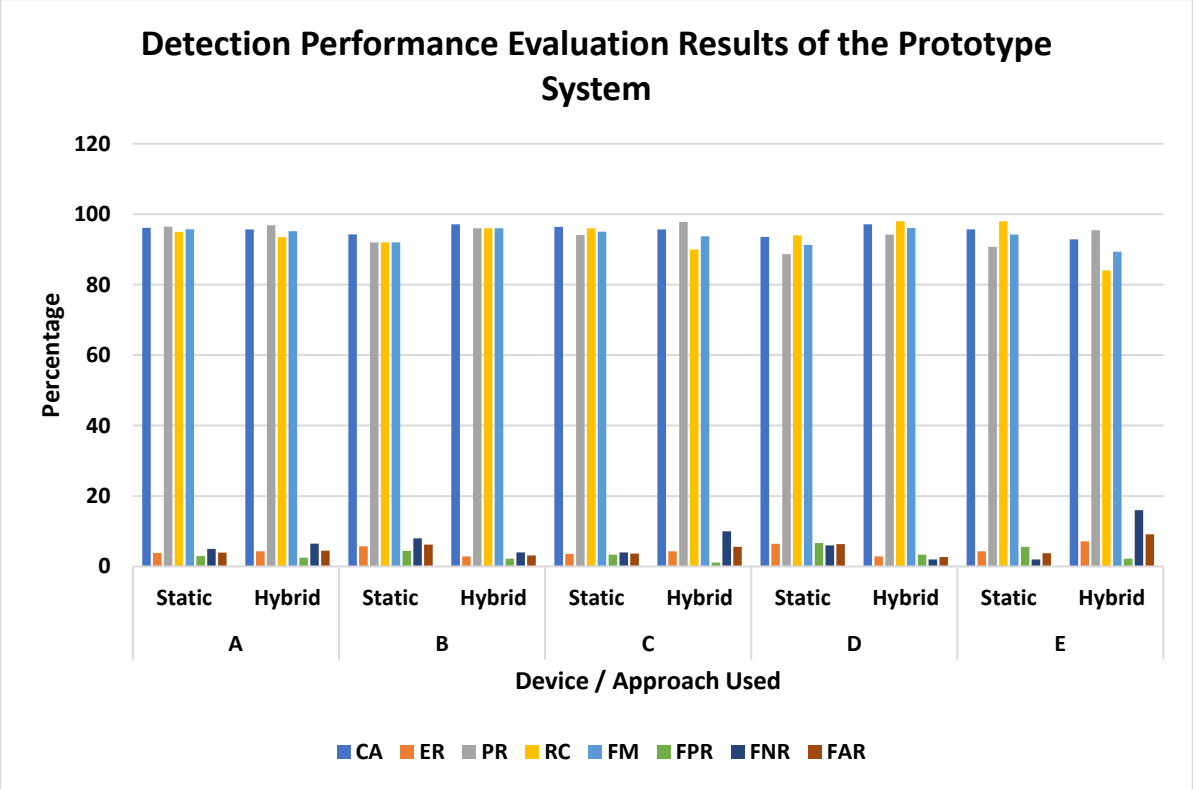

Figure 6.3 Detection Performance Evaluation Result

6.1.6 Energy Consumption Evaluation Of The Prototype System

The evaluation of energy consumption is necessary for this study because of the resource constraint nature of MDs in the MCC environment. During the experiment, the battery consumption of each device was recorded. The results of the energy consumption of each device, both when the prototype system was installed and when the prototype system was uninstalled from each device were recorded. The energy consumption of each device recorded during the experiment are presented in Table 6.7.

Table 6.7 Energy Consumption of each Device

DEVICE	Energy Usage with MINDPRES (%)	Energy Usage without MINDPRES (%)
A	78	71
B	86	79
C	75	64
D	83	71
E	89	75

The energy consumption was also evaluated using a standard benchmark tool to evaluate the energy consumption. This study uses the Android profiler as a benchmark to evaluate the energy performance of the prototype system. The Android profiler does not directly measure energy consumption. Rather, it uses a model that estimates the energy consumption for each resource on the device. This tool is integrated into the development environment and gives a detailed insight of

energy consumption of both simulated and real-life devices. In addition, other studies in this field have also adopted this tool for evaluation of energy consumption by Android apps (Gaska et al,2018; Farooq et al, 2019; Myasnikov et al,2021). The result of the Android profiler is also comparable to the energy consumption of the actual real-life device used for the experiment. Figure 6 in Appendix B shows the energy estimation level of the emulator used during development. The energy consumption evaluation of this study focused on the battery usage by the prototype system during the evaluation process (Table 6.7). The battery usage of device E was very high because of the types of apps installed on the device. On the device, dating apps and sports apps were installed which had high energy requirements due to the number of network activities in this apps category. The results presented in Figure 6.4 show that the prototype system consumes a considerable amount of energy during the experiment for a period of two weeks. Based on the results obtained during the prototype evaluation, one can conclude that the prototype system is feasible to be deployed on any Android device used in the MCC environment.

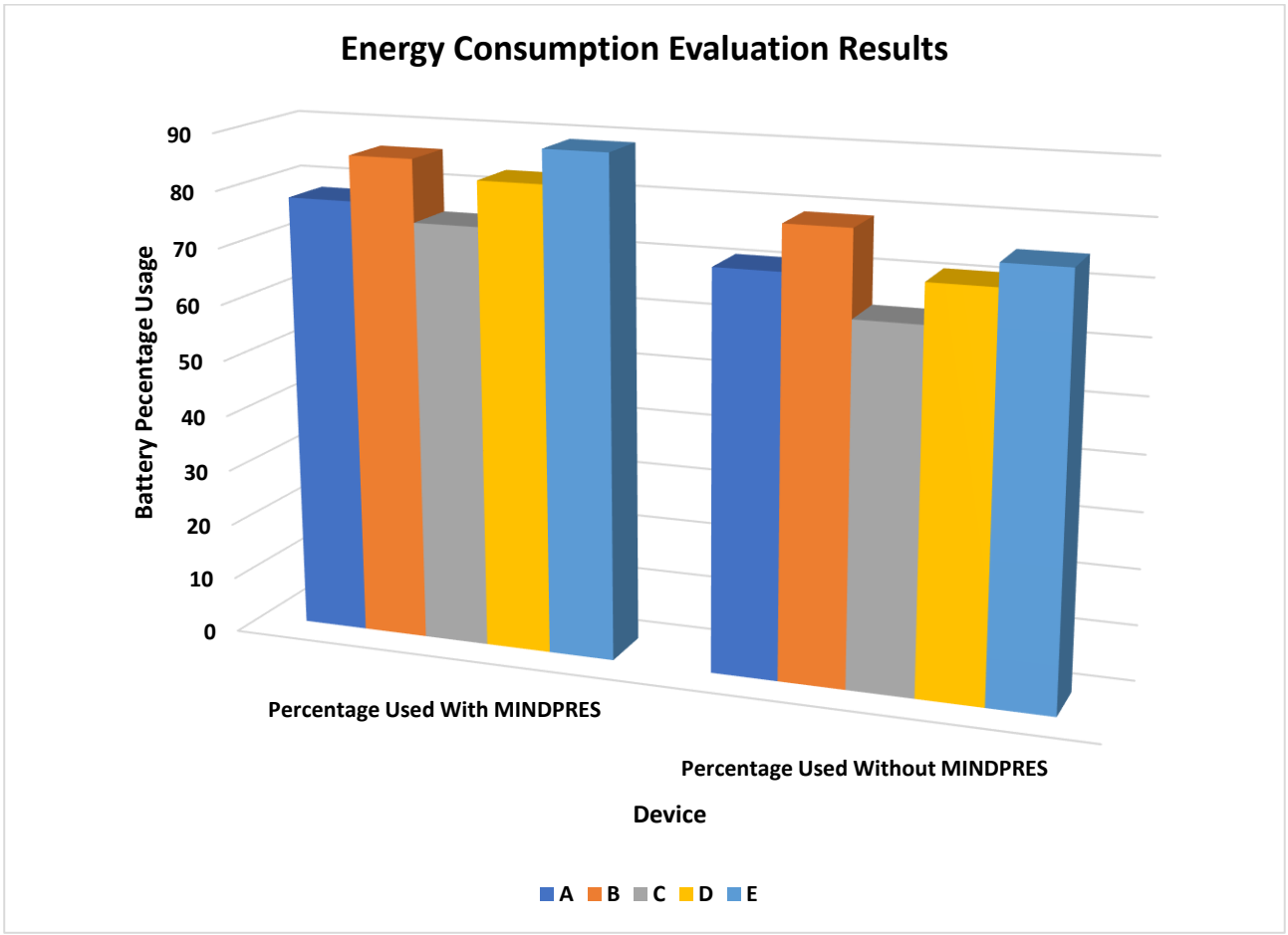

Figure 6.4 Energy Consumption Evaluation Result

6.2 PROTOTYPE SYSTEM EXPERT EVALUATION FEEDBACK

In this study, expert evaluation was carried out by three industry experts; two of them are employed in IT security roles, and one is an IT security consultant. Expert 1 (Osama Al Omari) is currently a security architect at New Zealand Post with over fifteen years' experience working in IT security roles in various companies. Expert 2 (Paul Hayes) has over ten years' experience as an IT security consultant to various NZ companies and Expert 3 (Eghbal Ghazizadeh) has over five years' experience in IT security roles. He is currently working as the Group Information Security Manager at Mercury NZ. The experts were approached based on the recommendation of the supervisory team for this research. The expert evaluation carried out by the invited experts covers one month period. This enables the expert to evaluate all the components that make up the prototype system. The invited experts evaluate the functionality and useability of the prototype system. After completing the expert validation process, each expert sends a report via email regarding the functionality and usability of the prototype system which are presented in the following sub sections. The contents presented in sections 6.2.1, 6.2.2 and 6.2.3 are the experts exact words contains in their email feedback as regards the evaluation of the prototype system.

6.2.1 EXPERT 1 FEEDBACK

I have read the documentation of this prototype system (MINDPRES) and tried it in action. I think the proposed prototype is very good in terms of detecting and preventing malicious applications. There is room for enhancement and coverage to do more practical threat detection and protection capabilities. Due to the nature of today's continuous threat landscape, more endpoint protection systems are tending to achieve more detection and prevention through behavioural analysis and more inspection of critical OS system calls and user land functions.

As an example, I attempted to install a malicious application, however the MINDPRES application stopped it, however I managed to run a reverse shell that loads a session and i was successful.

6.2.2 EXPERT 2 FEEDBACK

A. General Comments

1. It was interesting to see the level of background activity taking place when the device wasn't being used. Some logging and reporting capability would be a useful addition to the app, e.g., to report on for which apps there has been activity over the last hour even though the device and/or those apps haven't been used over that period.
2. It is important to consider what the information presented means to the average layman, as opposed to someone who has specialist knowledge in this field (e.g., when clicking down into the detail of a malicious activity, what does this mean to the average user?).

3. When apps have been identified as malicious or blacklisted, it should be clearly communicated what action has been taken by the MINDPRES app and what action is recommended to be taken by the user. This didn't appear to be clear from using the app.
4. Sometime the App locked up and needed to be restarted. I wasn't able to ascertain all the conditions that this occurred for and this may need further testing.
5. What information gets automatically or manually refreshed also needs to be considered as there are examples below of where this may have led to inconsistencies being displayed.
6. To fully evaluate functionality ideally needs the development of the Prevention Module (Blacklisted Activities) to be completed.

B. Device Manager – Specific Comments

1. The Device Manager appeared to be just an information page with no options to manage anything. If it doesn't manage anything (e.g., no settings or options to select) it should perhaps be renamed.
2. States the number of installed apps – this didn't automatically update when apps were either installed or removed.

C. App Evaluator – Specific Comments

1. Apps were easy to uninstall from the App Evaluator, although as noted earlier, the count of Apps installed on the Device Manager did not automatically update.
2. When a new app was added, I needed to disconnect and reconnect to Wi-Fi to see the new app listed. It would be better if this information was automatically updated.
3. The App Evaluator showed the 'Google Play services for Instant Apps' app as malicious and medium risk. For the uninformed user, this may raise some concerns as this is the app that is used to install other apps. This may be correct, but some information on why an app has been assessed as malicious would be useful.
4. The 'Video Call: Dating' app was mostly categorised as Malicious and Medium Risk, but on one occasion was shown as Benign and Low Risk (when MINDPRES was being used over a similar time period). This indicated some possible inconsistency.
5. com.zynga.wwf3.Words3Application was listed in the App Evaluator, but unlike for the other apps, selecting it didn't take you to the underlying list and the 'Uninstall button'.

D. Detection Engine – Specific Comments

1. As stated above, interesting to see the activities counts increase over a period of time (e.g., the last hour) when the device wasn't being used – indicating a level of background activity taking place over the network.
2. The count of activities for each page appeared to need to be refreshed manually and for each page, sometimes leading to inconsistencies between the number of activities displayed for each page, e.g., count of online activities showed as 17 whereas count of blacklisted activities showed as 18 – probably due to one of the pages not having been refreshed?
3. The Online Activities and Blacklisted Activities appeared to display the same information (if both were refreshed). Is this correct? Should all Online Activities be blacklisted? I assume this is due to this component still being under development.
4. On one occasion there was a notification that there were 5 malicious activities detected, but in the Malicious Activities page it showed only 4 malicious activities (this may have been due to a screen/page not being refreshed?).
5. When selecting a Malicious Activity or Blacklisted Activity for more information, the MINDPRES app sometimes locked-up – i.e. it said MINDPRES isn't responding and necessitated closing the app and re-launching.

6.2.3 EXPERT 3 FEEDBACK

It is my pleasure to review the MINDPRES mobile APP, I have worked with this app and overall, in two weeks, I have found that this Mobile Cloud Computing (MCC) consists of solutions that protect Android devices. I have found that this app employs different techniques and collect and analyse indicators of compromise to identify anomalous behaviour and counter threats. This App gathers data from the Android device/s and well as from external sources. Moreover, in my testing, I have found that the app works in conjunction with the device manager by leveraging the Android VPN services to monitor the behaviours of all apps in the device. As a summary, the app analyses all installed apps that have been installed on an Android device and determine the risk levels. My main recommendation is to focus of UX (https://www.total.com/designers/ux/mobile-ux-design-best-practices) and tuning the data specifically ML model.

6.2.4 EXPERT FEEDBACK SUMMARY

Overall, the feedbacks from all experts were positive and they all agreed that the prototype system is very effective in detecting Android malicious apps used in the MCC environment. However, each expert suggested areas for further improvement, as contained in the expert's summary report. Some of the feedbacks received from the expert was used to improve the implementation of the prototype

system. However, some of the recommended feedbacks from the invited security experts are beyond the scope of this study.

6.3 Results Comparison with Related Works

The results obtained from both the laboratory experiment discussed in Chapter Four and the real-life experiment results obtained from evaluating the prototype system in Table 6.6 are compared with other related malware detection systems in the extant literature. This is because the prototype system was implemented as a proof of concept to enhance data security in the MCC environment by focusing on the user layer of the MCC architecture. On the other hand, the implemented prototype system targets the Android OS environment. Therefore, comparing the results obtained in this study with the state-of-the-art Android malware detection system is necessary to evaluate the model's performance. The summary of the results from selected research articles published in peer-reviewed journals and conferences alongside the results recorded in this study is shown in Table 6.8.

It is evident from the results presented in Table 6.8 that most research work focuses only on permissions requested by an app as a feature to build a malware detection system using the static analysis approach, even though some have combined permission with other related components as features for building a more reliable detection model. This study has explored the combination of permissions, intent, and API requests by an app in a hybrid approach to analyse apps' static and dynamic behaviour in a hybrid approach. This study also considers monitoring app activities both when the user is using the device and when the device is not being use, to build a prototype system that can combat the threats caused by malicious apps that reside on the MCC users' devices.

In the literature, only a few works have proposed a solution to tackle data security issues caused by malicious apps executed by MDs in the MCC environment. For example, the work reported in OS (2021) proposed an intelligent model to combat threats caused by Android apps in the MCC environment using only permissions as a feature to build an ontology-based model. This study has combined permissions with intents and APIs to monitor apps. Apps are monitored using a VPN service approach combined with a hybrid analysis approach that alerts the user to the risk score and category of the apps on their device. This study complements the existing approach of many related works by monitoring the actual execution of apps on the cloud server by listening to the APIs and URL requests that the apps make in real-time. The prototype system implemented in this study, automatically analysed each request made in real-time using an ensemble ML model built using network traffic data obtained from the experiments conducted in this research.

Furthermore, the results presented in Table 6.8 show that most of the existing research in the literature has used only a few malware samples (less than 10,000 apps) to build a detection model, except for the work reported by Mathur et al. (2021) and Alazab et al. (2020). This study uses over 18,000 malicious app samples to build the detection model used to implement the prototype system.

This is because of the repositories (AndroZoo and RmvDroid) used for the data collection of apps in this study. The AndroZoo is currently the largest repository of Android malware samples in the research community, with over one million apps (both benign and malicious apps) APKs in their repository.

The detection performance results shown in Table 6.8 show that the models built in this research work's experimental and prototype implementation stages compare favourably with other related research. The results also improve the state-of-the-art solution proposed to combat malicious apps in the MCC environment and the mobile ecosystem. The experimental results, real-life and expert evaluation of the prototype system using apps in the official Android app stores show that the security solution proposed in this study is very effective in tackling the security issues caused bsy malicious apps in the MCC environment. The results show a classification accuracy of over 97% and a false positive rate of less than 2% in both laboratory and real-life experiments compared to other related works with less than 97% classification accuracy and a high false alarm rate of over 4%. Finally, some of the related works in the literature require root-level access to monitor apps' behaviour dynamically, but the prototype system implemented in this study does not require root-level access; instead, it leverages the Android VPN services to monitor the apps' behaviours in the MCC environment.

Table 6.8 Results Comparison with Related Works

Source	Features	Approach	Dataset	Size	MLC	Results
Proposed - Lab Experiment	P, I, A	Hybrid	AZ, RD	B-9,879 M-18,427	ECV	CA 98.16%, PR 98.95% RC 98.20% FM 98.57% FPR 1.90%
Proposed - Real-Life Experiment	P, I, A	Hybrid	AZ, RD, CMD2020 & GP	B-9,879 +600 M-18,427+400	ECV	CA 97.14% PR 97.83% RC 98.00% FM 96.08% FPR 1.11%
OS (2021)	P	Static	AZ, VSH	B-1,959 M-2,113	RF	CA-94.11% FM-93.00% FPR-3.00%
Cai et al (2021)	P, I, AC	Static	DB, AMD, GP	B-3,000 M-3,000	KNN	CA-96.58% PR-96.94% RC-96.47% FM-96.70%
Lu et al (2021)	P	Static	VT, DB	B-2,000 M-2,000	DNN	CA-95.83% PR-95.24% RC96.15% FM-95.69%
Mathur et al (2021)	P	Static	AZ	B-14,630 M-14,700	RF	CA- 96.95% FPR-3.32%
Alazab et al (2020)	P, A	Static	AZ	B-14,172 M-13,719	RF	FM-94.3%
Ribeiro et al (2019)	DR	Dynamic	GP	B-6,000 M-6,000	DT	CA-99.80%
Zhou et al (2019)	SC	Dynamic	BM, VSH	Not Reported	MCA	CA-97.85% PR-98.70% FPR-4.21%
Li, et al (2018)	P	Static	GP, Anzhi	B-5,494 M-2,650	DT	CA-93.62%
Idrees, et al (2017)	P, I	Static	GP, CT, DB, Gen	B-445 M-1300	ECV	PR- 98.40% FPR-0.10%
Feizollah, et al (2017)	P, I	Static	GP, DB	B-1,846 M-5,560	NB	PR-95.50% FPR-4.40%
Hatcher et al (2016)	P, SC	Static	Gen	Not Reported	DT	PR-94.59%
Hou, et al (2016)	SC	Dynamic	Not Reported	Not Reported	DL	CA-93.68%
Arp, et al (2014)	HC, P, I, A, NT	Static	GP, ACH	B-123,453 M-5,560	SVM	CA-94%
Saracino, et al (2014)	SC, P, A, UB	Hybrid	Gen, CT, VSH	B-9,804 M-2,800	KNN	PR-96.90%
Qi et al (2014)	NT	Dynamic	Gen	M-1,260	NB	CA-90.00%

Note: Permissions (P), Intent(I), API-(A),App Component(AC), Hardware Component (HC), Network Traffic (NT), Device Resource (DR), User Behaviour (UB),System Calls (SC),Benign (B) Malware (M),Decision Tree,(DT), Naïve Bayes (NB), Logistic Regression (LR), Random Forest (RF), KNearest Neighbour (k-NN), and Support Vector Machine, Monte Carlo Algorithm (MCA), Deep Learning (DL),Deep Neural Network (DNN),Ensemble Voting classifier (EVC),Classification Accuracy (CA), Precision Rate (PR), Recall Rate (RC), False Positive Rate (FPR), F-Score Measure (FM), AndroZoo (AZ), RmvDroid, (RD), CICMalDroid2020 (CMD2020), VirusShare (VSH), DREBIN (DB), Google Play (GP), VirusTotal (VT), Argus Lab's Android Malware Database (AMD), Baidu Mobile (BM), Anzhi (Anzhi), Contagio (CT), Genome (Gen), App China (ACH), Machine Learning Classifier (MLC**)**

6.4 CHAPTER SUMMARY

A detailed prototype system evaluation was carried out in this chapter. The prototype system implemented in this study can be built into the existing Android OS system and serve as a utility tool to enable MD users in the MCC environment to effectively secure their devices against threats caused by malicious apps that resides on their device. Notably, the prototype system implemented in this study logs all apps activities which can be useful for digital forensic investigation. The details of the logs are offloaded to the cloud server for future reference this also helps to prevent the MD user from tampering with the activity logs store locally in the devices since copies of this files are move periodically to the cloud to reduce the storage resources consumed by the prototype system.

To the best of my knowledge, there are no comparable real-life systems that target the MCC environment. However, relevant models have been proposed in the extant literature; future implementations may be compared to the prototype system developed in this study

However, the prototype evaluation was limited to only five real-life Android devices. This was because of the limited number of devices available to be borrowed from the university resource centre as at the time the prototype evaluation was carried out.

The evaluation results show that the model implemented as a proof of concept can effectively detect malicious activities of apps that reside on the MDs of MCC users. MINDPRES also compared favourably with existing solutions proposed in the extant literature as reported in this chapter. The permission risk value used in building the app evaluator that determines each app's riskiness was also presented in this chapter. This chapter also presents a detailed real-life experiment conducted using a new dataset different from the original dataset used for training the ensemble ML model used to implement the prototype system. This enables the model's evaluator to cope with new apps that will be designed over time and with the ability to detect malicious apps that can cause a zero-day attack in the MCC domain. The evaluations show a better detection rate and compare favourably to reported results. Overall, the prototype system implemented in this study was evaluated by invited security experts in the New Zealand IT industry to get their expert opinion. The feedback from the experts was very positive. However, the experts made a few suggestions to make the prototype system a great tool used by both the academic community and the industrial environment.

CHAPTER SEVEN

DISCUSSION AND CONCLUDING REMARKS

This study develops and proposes a novel ML-based framework that enhances the security of user data in the MCC environment. The research goal and objectives as defined in chapter one was accomplished at the end of the study. An instance of the framework was designed as a proof-of-concept prototype that addresses security issues caused by malicious apps residing on the MD nodes. The prototype system was developed and implemented for devices using the Android mobile OS. The ML model considers app permissions, intents and network activities (such as API calls at run-time) to identify potentially malicious activities of mobile apps and alerts the MD user who used their judgement to grant or deny the permissions requested by the app.

7.1 OVERVIEW OF THE STUDY

This study identifies gaps in existing security research in the MCC environment and proposes a novel framework that enhances the security of user data in the MCC environment. APK files of Android apps were collected from two repositories (AndroZoo and RmvDroid). The APK files were used to construct five different datasets used in ML experiments. The first experiment was carried out with ten ML classification algorithms using the first three datasets constructed in this study (datasets 1, 2 and 3) to identify the best performing ML algorithms with the constructed dataset using all the features (permissions and intents demanded by each app APK).

The second experiment involves the reduction of the feature sets used in experiment 1 by using a proposed filter-based FS statistical approach to select relevant features required to train an ML model. The selected features (permissions and intents) were used to construct dataset 4 from dataset 3. The reduced dataset and the best three ML classifiers (C1, C2 and C7) outcome of the first experiments were used in the development of an ensemble ML model using the static analysis approach. The ensemble ML model uses the selected features (permissions and intents demanded by an app) to distinguish malicious apps from benign apps.

The third experiment uses dataset 5 constructed from actual permissions and intents required at run time and network-related activities performed by each app using the dynamic analysis approach. The dynamic analysis approach monitors the actual behaviour of both malicious and benign apps on the device. The dynamic analysis approach analyses the network activities of each app using an Android emulator in a controlled environment to avoid infection of the university network. The experiment was carried out using a virtual machine that is isolated from the university network. The dynamic analysis approach also used the requested permissions and intent at run-time rather than the permissions and intent that were declared to be used by the app. This approach reveals the actual behaviour of the apps in the third experiment rather than the intended behaviour using the

static analysis approach. Both ensemble ML models using static and dynamic analysis approaches were developed and deployed to a cloud-based service for use in the prototype implementation.

The deployed ensemble ML models were used to implement the prototype system as a proof of concept. The prototype system was evaluated by invited security experts, and its performance was also evaluated using the confusion matrix alongside the energy consumption of the prototype system. The results of the evaluation show the system is effective at tackling data security issues in the user layer of the MCC architecture

7.2 Addressing the Research Questions

The main research goal of the study was to investigate how to protect MCC resources against attacks and enhance the security of user data in the MCC environment. The development of the novel framework and the implementation of the prototype system (MINDPRES) aimed to answer the main research question that guided the study and to achieve the specific research objectives of the sub research questions formulated in Chapter 1.

Main Research Question: What security components are required in a framework that can be used to protect MCC resources against attacks and enhance the security of user data in the MCC environment?

To answer the main research question, the following sub questions were formulated in this study

RSQ1: Which specific MCC resource require to be protected to enhance the security of the MCC environment?

RSQ2: What approach can be used to protect the identified MCC resource in RSQ1?

RSQ3: What metrics can be used to evaluate the performance of the approach identified in RSQ2 above and how can this approach be implemented to protect the relevant MCC resource?

Overall, the research goal set in at the start of the study, was successfully met. As discussed in Chapter 2, a comprehensive literature review was conducted to answer the first research sub-question. An in-depth study of existing security frameworks that offer protection to the MCC resources was carried out. The frameworks proposed in the literature were analysed using the MCC security requirements and the egregious eleven threat model (Kissel, 2011; Liu et al., 2011; Mogull et al.,2017; CSA, 2019). Based on the analysis of the results reported in the extant literature it was concluded that despite the significant vulnerability level of the MD user layer of the MCC environment, research that has been carried out to address these security issues associated with the user layer of the MCC architecture was relatively scarce.

The further study of MD security indicated that attackers had found a way around developing malicious apps that were able to avoid detection by most existing detection techniques. In addition,

the security vulnerabilities associated with the popular Android mobile OS (installed in many of the MCC user devices) have contributed to the recent growth in the development of malicious apps able to be uploaded to the Google play store. The Android OS does have system and application security countermeasures that offer protection to its users. However, the Android OS application security control layer of the Android OS is built on a permission-based model that generally relies on the end-user to judge whether an app legitimately requires (or not) certain permissions without knowing the intention of the app developer. This weakness in the Android OS used in the MCC environment has attracted malware developers to target these devices to obtain sensitive information that can compromise both the mobile and cloud environment in the MCC domain. Therefore, in this study, the MD user layer was identified as the specific MCC resource that requires adequate protection to enhance the security of this environment.

To answer the second research sub-question, the initial results of the comprehensive literature review of the security frameworks that target the MCC environment were analysed further to identify the best approach to protect the MD as a resource in the MCC environment. The protection methods used by each framework were evaluated based on the number of threats each framework offers protection against, using the egregious eleven threat model proposed by the Cloud Security Alliance in 2019 as a benchmark (CSA, 2019). The results showed that most of the studies included in the literature applied cryptographic and biometric authentication models to combat the threats that faced the MCC environment. However, these approaches provide protection against only a few of the known threats in the MCC domain.

This study aimed to develop a framework that offered a better the protection of the MD resources in the MCC environment. This necessitates the requirement to provides protection against a higher number of threats in the MCC environment compared to the protection given by the cryptographic and biometric authentication approaches that are predominantly used in current research.

As shown by the outcomes of the analysis of the existing frameworks, only a few included IDS as an integral part of the protection approach. However, the IDS-based frameworks were found to be performing better than other proposed approaches regarding the number of threats they offered protection against. The identification of IDS as a better approach necessitated another comprehensive review of work that applied IDS as a security technique in the MCC, CC, and MC environments. The review showed that IDS applying ML techniques provided a significantly better protection against the egregious threats (as the benchmark for this study) compared to other approaches. Hence, this study answers the second sub research question by proposing a novel IDPS approach that uses the ensemble ML techniques named MINDPRES (Mobile-Cloud Intrusion Detection and Prevention System). The proposed approach uses hybrid (static and

dynamic analysis) analysis of device behaviour to protect MD users from malicious apps in the MCC environment.

To address the first part of the third research sub-question and identify the metrics to be used in the evaluation of the efficiency of the proposed approach, a detailed review of the literature of existing works that used ML classification models for IDS and IDPS as an approach for security in the MCC and CC domains was conducted first. The analysis of the results showed that most studies had adopted the confusion matrix to derive a metrics for the evaluation of the intrusion detection models' performance. In all studies, classification accuracy was used as the most important metric and an evaluation criterion. In addition, most of the studies also considered the false alarm rate as an evaluation criterion and a high rate of false alarm was reported as one of the performance issues with ML-based IDS. Other evaluation criteria's, such as precision rate, recall rate, and F-score measure values, were also used but only in a few of the studies reviewed.

In this study, the evaluation metrics used for the evaluation of the proposed prototype system's performance use a confusion matrix that consider malware detection output. In addition, most of the studies included in the discussion did not consider the evaluation of the energy consumption of the MD nodes in the MCC domain considering the resource constrained nature of these devices. This is another critical issue that was identified during this research. Therefore, this study uses the energy consumption level of the various devices, classification accuracy, precision rate, recall rate, f-measure scores, false alarm rate, false positive rate, and false negative rate as metrics to evaluate the performance of the prototype system (MINDPRES).

To address the second part of the third research sub question, a set of malicious and benign app APKS samples collected from the AndroZoo and RmvDroid repositories was used to build an ensemble ML models. (Developed and discussed in Chapter 4). The ensemble ML model was implemented in a proof-of-concept prototype system (MINDPRES), presented, and discussed in Chapter 5. The performance of the prototype system was evaluated using the metrics identified above; the results were discussed in Chapter 6. In addition, a group of invited New Zealand based IT security experts working in related industry sectors provided evaluation reports considering the features and performance characteristics of the prototype system as installed and activated. The performance evaluation outcomes and the experts' reports indicated that the prototype system implemented as a proof of concepts was able to effectively detect intrusions in the MCC environment caused by malicious apps at the MD nodes.

Finally, to answer the main research question, after providing answers to the sub research questions. The novel framework proposed in this study has identified the following security components as being required in a framework to protect MCC resources against attacks and improve the security of user data in this environment:

A. An IDPS with ensemble ML models using static and dynamic analysis of device behaviour can protect resources in both mobile and cloud environment.

B. A strong cryptographical system to protect leakage of sensitive information when mobile apps are executed in the MCC environment both at rest and in motion.

C. A-hypervisor-based IDPS to protect the cloud infrastructures against insider attacks.

D. A strong identity and trust management security components to combat illegal access to cloud resources of MCC users.

Furthermore, it was proposed that comprehensive framework for the protection of MCC resources should address at least five or more top security threats that cut across the different layers of the MCC architecture. It was suggested in particular that threat types T1, T4, T5, T7, T9, and T10 should be addressed at the MCUL (user device) layer, threat types T1, T2 and T8 at the MNCL layer, and threats type T1, T2, T3, T4, T5, T6, T7, T10 and T11 at the MCSPL layer. The ML-based protection system developed, implemented, and evaluated in this study addresses the data security issues related to use of malicious apps which is, one of the topmost threats facing users of MCC infrastructure. Malicious apps present dangerous threats, affecting both mobile and cloud environments' security. In particular, threats T1, T5, T7, T10, and T11 that exploit vulnerabilities at the MCUL layer of the MCC environment. The system is highly relevant to current cybersecurity environment for example, the prototype system that was implemented in this study (MINDPRES) addresses four of the six most important threats (namely malicious apps and websites, mobile ransomware, phishing, and advanced jailbreaking and rooting techniques) identified in the report published by Checkpoint (Checkpoint, 2021). The prototype system developed and implemented in this study only address the security component of the novel framework in A above to tackle data security issues at the user layer of the MCC environment. The other security components identified in B, C and D are part of the proposed novel framework that addresses other security issues in other layers of the MCC environment.

7.3 RESEARCH CONTRIBUTION

The process involved in the development and evaluation of a novel framework to improve the data security in the MCC environment in this study has revealed some of the major contributions of this research. It was evident that the literature review analysis approach of related works that was used in this study, helped identify the research gaps. The adoption of the DSRM resulted in the design of a prototype system (artifact) that can be integrated into existing mobile OS as a utility tool to protect MCC user against the vulnerabilities and threats caused by malicious apps. In addition, the following are some of the contributions of this research to the body of knowledge.

A. This study proposes a novel security framework which identifies the security threats that needs to be addressed at the different layers of the MCC architecture to provide a more comprehensive solution for protection of MCC resources. These requirements may serve as

guideline for both organizations and MCC service providers for developing relevant security programmes for their services and cloud Infrastructure.

B. The study proposes a novel approach towards controlling app behaviour by empowering the user to make an informed judgement. The system monitors all network activities of mobile apps within the device without root level access and automatically blocks the detected malicious apps activities and provides the MD user with the option to enable the activities in the event of false alarm. This is important, because the system mitigates the negative effect of potentially false alarm rate and allows the app to operate freely if an app activity has been determined as not malicious.
C. The proposed cloud-based approach for risk assessment by the app evaluator to determine the risk score and category of mobile apps at the MD layer using ensemble ML techniques and a novel statistical approach with dangerous permissions and intents frequency usage by both malicious and benign apps is the first of its kind in the MCC domain to the best of my knowledge.
D. The development of a filter-based FS technique using a statistical approach (based on the frequency of usage of permissions and intents by both malicious and benign apps) to select relevant features used in the development of the ensemble ML model.
E. The proposed ML-based IDPS approach using ensembling techniques in the study is the first of its kind in the MCC domain to the best of my knowledge. Only in a few works an IDS have been used in the MCC environment. Furthermore, most such work has not really focused on the MD user layer in the MCC environment. Also, no study in MCC to the best of my knowledge has combined both static and dynamic analysis of device behaviour at run time with user activities. MINDPRES combines static and dynamic analysis of device behaviour with the ML technique for protection of MCC resources against attacks.

7.4 Challenges and Limitations of the Study

In this study, different challenges were encountered as regards the implementation of the novel solution to tackle security issues faced by the user layer of the MCC architecture. First, issues with data collection arise at the time of the development of the ML model. There was no readily available dataset that contained the features required in this study for the development of the ML model. Similarly, there are no publicly available repositories that contain app installation files for other mobile platforms, such as iOS. Hence, the prototype system implemented in this study can only work on Android mobile devices because the dataset constructed for the development of the ensemble ML model used by the detection engine and the app evaluator of the prototype system was constructed from the Android APK files collected from different repositories. These challenges limit this study's prototype implementation to the Android mobile platform. An important lesson learnt was that constructing one’s own dataset was a laborious and time-consuming process that needed to be carried out rigorously to ensure the credibility of the subsequent experiments.

In addition, due to the possibility of infection of the university network, the malicious APK sample files were stored and processed in an isolated environment different from the university network. Due to the storage and processing capability of the resources available for this study, only 40,000 APK files (benign and malicious app samples) were initially collected and processed. Hence, this study uses only 28,306 Android APKs for its training and testing during the laboratory experiment. Another lesson learnt was that the dataset construction needed to be supported by adequate resources acquired at the preparation as not to impose additional limitations of the study.

Finally, the testbed set used for the evaluation of the prototype system was only limited to 1,000 apps because of the storage space and memory capability of the Android tablets devices used for the evaluation of the prototype system. The malicious app samples used for the evaluation require root-level access to the devices. Hence, the evaluation of the prototype type system did not use any personal devices for actual, real-life experiments. This study is limited to the evaluation of apps that reside on university devices and do not contain personally identifiable information about any individual. While the use of devices containing personal data would not be justifiable for the purposes of this study a 'real-life' testing may extend the use of a large number of apps in the prototype evaluation may show a more realistic detection performance compared to what was obtainable in this study.

7.5 Directions for Further Research

The prototype was completed in 2021. While the security landscape may have changed, especially with the demands on MC and MCC during the global pandemic, the threats and attacks considered in this study are still relevant. The goals and objectives specified are met in this study as evidenced by the answers provided to the main research questions and the sub research questions. The proposed framework may be improved further by extending it to include digital forensics and other IDPS techniques that are not covered in this study. In addition, future work may address security issues at the mobile communication channel layers and the mobile cloud service provider layer of the MCC architecture. Other possible research directions are stated as follows:

A. Implementation and evaluation of the prototype system in another mobile OS environment different from Android.
B. Detection of malicious activities in the MCC environment by behavioural analysis approach and inspection of critical OS system calls and user land functions to improve the detection and prevention system.
C. Energy consumption optimization of the prototype system in Internet of Things (IoTs) devices.
D. Designing repositories for Installation files of other mobile platforms such as iOS so that researchers can also implement the prototype system in this environment

E. Tackling the issues of insider attacks in the service providers layer of the MCC environment.

F. Application of deep learning techniques to improve the detection engine performance

Finally, there is need for the development and implementation of other security component such as the strong cryptographical system to protect leakage of sensitive information when devices offload data to the cloud environment. The development of the hypervisor based IDPS to protect the cloud infrastructures against insider attacks. The development and implementation of strong identity and trust management security components to combat illegal access to cloud resources of MCC users.

REFERENCES

Achbarou, O., El Kiram, M. A., Bourkoukou, O., & Elbouanani, S. (2018). A Multi-agent System-Based Distributed Intrusion Detection System for a Cloud Computing. In *International Conference on Model and Data Engineering* (pp. 98-107). Springer, Cham.

Agrawal, N., & Tapaswi, S. (2019). A trustworthy agent-based encrypted access control method for the mobile cloud computing environment. *Pervasive and Mobile Computing*, *52*, 13-28.

Aguiar, E., Zhang, Y., & Blanton, M. (2014). An overview of issues and recent developments in cloud computing and storage security. High Performance Cloud Auditing and Applications, 3-33.

Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., & Awajan, A. (2020). Intelligent mobile malware detection using permission requests and API calls. Future Generation Computer Systems, 107, 509-521.

Alghofaili, Y., Albattah, A., Alrajeh, N., Rassam, M. A., & Al-rimy, B. A. S. (2021). Secure Cloud Infrastructure: A Survey on Issues, Current Solutions, and Open Challenges. Applied Sciences, 11(19), 9005.

Ali, M., Khan, S. U., & Vasilakos, A. V. (2015). Security in cloud computing: Opportunities and challenges. Information sciences, 305, 357-383.

Almorsy, M., Grundy, J., & Müller, I. (2016). An analysis of the cloud computing security problem. arXiv preprint arXiv:1609.01107.

Alomari, E., Manickam, S., Gupta, B. B., Karuppayah, S., & Alfaris, R. (2012). Botnet-based distributed denial of service (DDoS) attacks on web servers: classification and art. arXiv preprint arXiv:1208.0403.

AlShahwan, F., Faisal, M., & Ansa, G. (2016). Security framework for RESTful mobile cloud computing Web services. *Journal of Ambient Intelligence and Humanized Computing*, *7*(5), 649-659.

Alshehri, A., Hewins, A., McCulley, M., Alshahrani, H., Fu, H., & Zhu, Y. (2017). Risks behind device information permissions in Android OS. Communications and Network, 9(04), 219.

Alshehri, A., Marcinek, P., Alzahrani, A., Alshahrani, H., & Fu, H. (2019). Puredroid: Permission usage and risk estimation for android applications. In Proceedings of the 2019 3rd International Conference on Information System and Data Mining (pp. 179-184).

Allix, K., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2016). Androzoo: Collecting millions of android apps for the research community. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR) (pp. 468-471). IEEE.

Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. Journal of Computational Science, 25, 152-160.

Al_Janabi, S., & Hussein, N. Y. (2019). The reality and future of the secure mobile cloud computing (SMCC): survey. In International Conference on big data and networks technologies (pp. 231-261). Springer, Cham.

Al_Janabi, S. (2020). Smart system to create an optimal higher education environment using IDA and IOTs. International Journal of Computers and Applications, 42(3), 244-259.

Al-Hemairy, M., Amin, S., & Trabelsi, Z. (2009). Towards more sophisticated ARP Spoofing detection/prevention systems in LAN networks. In 2009 International Conference on the Current Trends in Information Technology (CTIT) (pp. 1-6). IEEE.

Android O.S (2021). Security Documentation: https://source.android.com/security

Anwer, H. M., Farouk, M., & Abdel-Hamid, A. (2018). A framework for efficient network anomaly intrusion detection with features selection. IEEE. Symposium conducted at the meeting of the 2018 9th International Conference on Information and Communication Systems (ICICS)

Ambusaidi, M. A., He, X., Nanda, P., & Tan, Z. (2016). Building an intrusion detection system using a filter-based feature selection algorithm. IEEE transactions on computers, 65(10), 2986-2998.

Arabo A., Pranggono, B., (2013). Mobile Malware and Smart Device Security: Trends, Challenges and Solutions, in Control Systems and Computer Science (CSCS), 2013. In: Proceedings of the 19th International Conference on, pp. 526–531.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014, February). Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26).

Arvind, K. S., & Manimegalai, R. (2017). Secure data classification using superior naive classifier in agent-based mobile cloud computing. *Cluster Computing*, *20*(2), 1535-1542.

Babu, B. M., & Bhanu, M. S. (2015). Prevention of insider attacks by integrating behavior analysis with risk-based access control model to protect cloud. Procedia Computer Science, 54, 157-166.

Ba, H., Heinzelman, W., Janssen, C. A., & Shi, J. (2013). Mobile computing-A green computing resource. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 4451-4456). IEEE.

Bahrami, M., & Singhal, M. (2015). A light-weight permutation-based method for data privacy in mobile cloud computing. In 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (pp. 189-198). IEEE.

Balamurugan, V., & Saravanan, R. (2017). Enhanced intrusion detection and prevention system on cloud environment using hybrid classification and OTS generation. *Cluster Computing*, 1-13.

Barbhuiya, S., Kilpatrick, P., & Nikolopoulos, D. S. (2020). DroidLight: Lightweight anomaly-based intrusion detection system for smartphone devices. In Proceedings of the 21st International Conference on Distributed Computing and Networking (pp. 1-10).

Bedi, R. K., Singh, J., & Gupta, S. K. (2021). An efficient and secure privacy preserving multi-cloud storage framework for mobile devices. International Journal of Computers and Applications, 43(5), 472-482.

Belouch, M., El Hadaj, S., & Idhammad, M. (2017). A two-stage classifier approach using reptree algorithm for network intrusion detection. International Journal of Advanced Computer Science and Applications, 8(6), 389-394.

Benabied, S., Zitouni, A., & Djoudi, M. (2015). A cloud security framework based on trust model and mobile agent. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)* (pp. 1-8). IEEE.

Besharati, E., Naderan, M., & Namjoo, E. (2018). LR-HIDS: logistic regression host-based intrusion detection system for cloud environments. *Journal of Ambient Intelligence and Humanized Computing*, 1-24.

Bhattacharya, S., & Selvakumar, S. (2016). Multi-measure multi-weight ranking approach for the identification of the network features for the detection of DoS and Probe attacks. The Computer Journal, 59(6), 923-943.

Bostani, H., & Sheikhan, M. (2017). Hybrid of binary gravitational search algorithm and mutual information for feature selection in intrusion detection systems. Soft computing, 21(9), 2307-2324.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599-616.

Cai, L., Li, Y., & Xiong, Z. (2021). JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. Computers & Security, 100, 102086.

Çavuşoğlu, Ü. (2019). A new hybrid approach for intrusion detection using machine learning methods. Applied Intelligence, 49(7), 2735-2761.

Cen, L., Gates, C. S., Si, L., & Li, N. (2014). A probabilistic discriminative model for android malware detection with decompiled source code. IEEE Transactions on Dependable and Secure Computing, 12(4), 400-412.

Chaudhry, J. A., Chaudhry, S. A., & Rittenhouse, R. G. (2016). Phishing attacks and defenses. *International Journal of Security and Its Applications*, *10*(1), 247-256.

Chean, L. T., Ponnusamy, V., & Fati, S. M. (2018). Authentication scheme using unique identification method with homomorphic encryption in Mobile Cloud Computing. In *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)* (pp. 195-200). IEEE.

Checkpoint Security Report (2021) https://www.checkpoint.com/cyber-hub/threat prevention/what-is-mobile-security/top-6-mobile-security-threats-and-how-to-prevent-them/#TopThreats

Chen, D., & Zhao, H. (2012). Data security and privacy protection issues in cloud computing. In 2012 International Conference on Computer Science and Electronics Engineering (Vol. 1, pp. 647-651). IEEE.

Chen, Y. J., & Wang, L. C. (2011). A security framework of group location-based mobile applications in cloud computing. In *2011 40th International Conference on Parallel Processing Workshops* (pp. 184-190). IEEE.

Chung, K. Y., Yoo, J., & Kim, K. J. (2014). Recent trends on mobile computing and future networks. Personal and Ubiquitous Computing, 18(3), 489-491.

Cushman, I. J., Al Sadi, M. B., Chen, L., & Haddad, R. J. (2017). A Framework and the Design of Secure Mobile Cloud with Smart Load Balancing. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)* (pp. 205-210). IEEE.

CSA (2013) The Notorious 9 - Cloud Computing Top Threats in 2013, Tech. Rep., Cloud Security Alliance, URL https://cloudsecurityalliance.org/artifacts/the-notorious-nine-cloud-computing-top-threats-in-2013/ [Accessed on 10-May-2019], 2013.

CSA (2016) The Treacherous 12 - Cloud Computing Top Threats in 2016, Tech. Rep., Cloud Security Alliance, URL https://cloudsecurityalliance.org/artifacts/the-treacherous-twelve-cloud-computing-top-threats-in-2016/ [Accessed on 10-May-2019]

CSA (2019) The Egregious 11 - Cloud Computing Top Threats in 2019, Tech. Rep., Cloud Security Alliance, URL https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/, [Accessed on 22-October-2019], 2019.

Dai, J., & Zhou, Q. (2010). A PKI-based mechanism for secure and efficient access to outsourced data. In 2010 International Conference on Networking and Digital Society (Vol. 1, pp. 640-643). IEEE.

Dai, Q., Yang, H., Yao, Q., & Chen, Y. (2012). An improved security service scheme in a mobile cloud environment. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems* (Vol. 1, pp. 407-412). IEEE.

Dbouk, T., Mourad, A., Otrok, H., & Talhi, C. (2016). Towards an ad-hoc cloud-based approach for mobile intrusion detection. In *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (pp. 1-8). IEEE.

Derhab, A., Belaoued, M., Guerroumi, M., & Khan, F. A. (2020). Two-factor mutual authentication offloading for mobile cloud computing. IEEE Access, 8, 28956-28969.

Dey, S., Sampalli, S., & Ye, Q. (2015). A context-adaptive security framework for mobile cloud computing. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)* (pp. 89-95). IEEE.

Dey, S., Ye, Q., & Sampalli, S. (2019). A machine learning-based intrusion detection scheme for data fusion in mobile clouds involving heterogeneous client networks. *Information Fusion*, *49*, 205-215.

Dhage, S. N., Meshram, B. B., Rawat, R., Padawe, S., Paingaokar, M., & Misra, A. (2011). An intrusion detection system in a cloud computing environment. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology* (pp. 235-239). ACM.

Dhanya, N. M., & Kousalya, G. (2015). Adaptive and Secure Application Partitioning for Offloading in Mobile Cloud Computing. In *International Symposium on Security in Computing and Communication* (pp. 45-53). Springer, Cham.

Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, *13*(18), 1587-1611.

Dolgikh, A., Birnbaum, Z., Chen, Y., & Skormin, V. (2013). Behavioural modelling for suspicious process detection in cloud computing environments. In *2013 IEEE 14th International Conference on Mobile Data Management* (Vol. 2, pp. 177-181). IEEE.

Donald, A. C., & Arockiam, L. (2015, January). A secure authentication scheme for MobiCloud. In *2015 International Conference on Computer Communication and Informatics (ICCCI)* (pp. 1-6). IEEE.

Eesa, A. S., Orman, Z., & Brifcani, A. M. A. (2015). A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems. Expert Systems with Applications, 42(5), 2670-2679.

Farooq, M. U., Khan, S. U. R., & Beg, M. O. (2019). Melta: A method level energy estimation technique for android development. In 2019 International Conference on Innovative Computing (ICIC) (pp. 1-10). IEEE.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: Analysis of android intent effectiveness in malware detection. computers & security, 65, 121-134.

Fernandes, D. A., Soares, L. F., Gomes, J. V., Freire, M. M., & Inácio, P. R. (2014). Security issues in cloud environments: a survey. International Journal of Information Security, 13(2), 113-170.

Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. Future generation computer systems, 29(1), 84-106.

Feng, Y., Chen, L., Zheng, A., Gao, C., & Zheng, Z. (2019). Ac-net: Assessing the consistency of description and permission in android apps. IEEE Access, 7, 57829-57842.

Ficco, M., Venticinque, S., & Di Martino, B. (2012). Mosaic-based intrusion detection framework for cloud computing. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*(pp. 628-644). Springer, Berlin, Heidelberg.

Fischer, A., Kittel, T., Kolosnjaji, B., Lengyel, T. K., Mandarawi, W., de Meer, H., & Weishäupl, E. (2015). CloudIDEA: a malware defence architecture for cloud data centres. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 594-611). Springer, Cham.

Gaharwar, R. S., & Gupta, R. (2020). Vulnerability assessment of android instant messaging application and network intrusion detection prevention systems. Journal of Statistics and Management Systems, 23(2), 399-406.

Gai, K., Qiu, M., Tao, L., & Zhu, Y. (2016). Intrusion detection techniques for mobile cloud computing in heterogeneous 5G. *Security and Communication Networks*, *9*(16), 3049-3058.

Gaska, B., Gniady, C., & Surdeanu, M. (2018). MLStar: Machine Learning in Energy Profile Estimation of Android Apps. In Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (pp. 216-225).

Ghribi, S., Makhlouf, A. M., & Zarai, F. (2018). C-DIDS: A Cooperative and Distributed Intrusion Detection System in Cloud environment. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*(pp. 267-272). IEEE.

Goyal, D., & Krishna, M. B. (2015). Secure framework for data access using Location-based service in Mobile Cloud Computing. In *2015 Annual IEEE India Conference (INDICON)* (pp. 1-6). IEEE.

Grobauer, B., Walloschek, T., & Stocker, E. (2010). Understanding cloud computing vulnerabilities. IEEE Security & privacy, 9(2), 50-57.

Gupta, B. B., Yamaguchi, S., & Agrawal, D. P. (2018). Advances in security and privacy of multimedia big data in mobile and cloud computing. Multimedia Tools and Applications, 77(7), 9203-9208.

Gupta, S., Horrow, S., & Sardana, A. (2012). IDS based defence for cloud-based mobile infrastructure as a service. In *2012 IEEE Eighth World Congress on Services* (pp. 199-202). IEEE.

Hashizume, K., Rosado, D. G., Fernández-Medina, E., & Fernandez, E. B. (2013). An analysis of security issues for cloud computing. Journal of internet services and applications, 4(1), 1-13.

Hatcher, W. G., Maloney, D., & Yu, W. (2016). Machine learning-based mobile threat monitoring and detection. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*(pp. 67-73). IEEE.

Hazarika, P., Baliga, V., & Tolety, S. (2014). The mobile-cloud computing (MCC) roadblocks. In *2014 Eleventh International Conference on Wireless and Optical Communications Networks (WOCN)* (pp. 1-5). IEEE.

Heninger, N., Durumeric, Z., Wustrow, E., & Halderman, J. A. (2012). Mining your Ps and Qs: Detection of widespread weak keys in network devices. In 21st {USENIX} Security Symposium ({USENIX} Security 12) (pp. 205-220).

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 75-105.

Huang, D., Zhang, X., Kang, M., & Luo, J. (2010). MobiCloud: building secure cloud framework for mobile computing and communication. In *2010 fifth IEEE international symposium on service-oriented system engineering* (pp. 27-34). IEEE.

Huang, D., Zhou, Z., Xu, L., Xing, T., & Zhong, Y. (2011). Secure data processing framework for mobile cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 614-618). IEEE.

Houmansadr, A., Zonouz, S. A., & Berthier, R. (2011). A cloud-based intrusion detection and response system for mobile phones. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)* (pp. 31-32). IEEE.

Hou, S., Saas, A., Chen, L., & Ye, Y. (2016). Deep4maldroid: A deep learning framework for android malware detection based on Linux kernel system call graphs. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)* (pp. 104-111). IEEE.

Idhammad, M., Afdel, K., & Belouch, M. (2017). Dos detection method based on artificial neural networks. International Journal of Advanced Computer Science and Applications, 8(4), 465-471.

Idrees, F., & Muttukrishnan, R. (2014). War against mobile malware with cloud computing and machine learning forces. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)* (pp. 278-280). IEEE.

Idrees, F., Rajarajan, M., Conti, M., Chen, T. M., & Rahulamathavan, Y. (2017). PIndroid: A novel Android malware detection system using ensemble learning methods. Computers & Security, 68, 36-46.

Idrissi, H., Ennahbaoui, M., El Hajji, S., & Souidi, E. M. (2017). A Secure Cloud-Based IDPS Using Cryptographic Traces and Revocation Protocol. In *International Conference on Codes, Cryptology, and Information Security* (pp. 365-382). Springer, Cham.

Inayat, Z., Gani, A., Anuar, N. B., Anwar, S., & Khan, M. K. (2017). Cloud-based intrusion detection and response system: open research issues, and solutions. *Arabian Journal for Science and Engineering*, *42*(2), 399-423.

Irshad, A., Chaudhry, S. A., Alomari, O. A., Yahya, K., & Kumar, N. (2020). A novel pairing-free lightweight authentication protocol for mobile cloud computing framework. IEEE Systems Journal, 15(3), 3664-3672.

Itani, W., Kayssi, A., & Chehab, A. (2010). Energy-efficient incremental integrity for securing storage in mobile cloud computing. In *2010 International Conference on Energy-Aware Computing* (pp. 1-2). IEEE.

Jensen, M., Gruschka, N., & Herkenhöner, R. (2009). A survey of attacks on web services. Computer Science-Research and Development, 24(4), 185-197.

Jia, W., Zhu, H., Cao, Z., Wei, L., & Lin, X. (2011). SDSM: a secure data service mechanism in mobile cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 1060-1065). IEEE.

Jing, Y., Ahn, G. J., Zhao, Z., & Hu, H. (2014). Towards automated risk assessment and mitigation of mobile applications. IEEE Transactions on Dependable and Secure Computing, 12(5), 571-584.

Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2016). Cypider: building community-based cyber-defense infrastructure for android malware detection. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (pp. 348-362).

Kaspersky Security Lab (2021) Mobile Malware Evaluation 2021, https://securelist.com/it-threat-evolution-in-q3-2021-mobile-statistics/105020/

Khan, A. N., Kiah, M. M., Madani, S. A., & Ali, M. (2013). Enhanced dynamic credential generation scheme for the protection of user identity in mobile-cloud computing. *The Journal of Supercomputing*, *66*(3), 1687-1706.

Khan, A. N., Kiah, M. M., Madani, S. A., Ali, M., & Shamshirband, S. (2014). Incremental proxy re-encryption scheme for the mobile cloud computing environment. *The Journal of Supercomputing*, *68*(2), 624-651.

Khatri, S. K., & Vadi, V. R. (2017). Biometric-based authentication and access control techniques to secure mobile cloud computing. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)* (pp. 1-7). IEEE.

Khedr, W. I., Hosny, K. M., Khashaba, M. M., & Amer, F. A. (2020). Prediction-based secured handover authentication for mobile cloud computing. Wireless Networks, 26, 4657-4675.

Kholidy, H. A., Erradi, A., Abdelwahed, S., & Baiardi, F. (2016). A risk mitigation approach for autonomous cloud intrusion response system. *Computing*, *98*(11), 1111-1135.

Khune, R. S., & Thangakumar, J. (2012). A cloud-based intrusion detection system for Android smartphones. In *2012 International Conference on Radar, Communication and Computing (ICRCC)* (pp. 180-184). IEEE.

Kim, H., Kim, J., Kim, Y., Kim, I., & Kim, K. J. (2018). Design of network threat detection and classification based on machine learning on cloud computing. *Cluster Computing*, 1-10.

Kim, S., Hwang, C., & Lee, T. (2020). Anomaly based unknown intrusion detection in endpoint environments. Electronics, 9(6), 1022.

Kim, K., Kim, J., Ko, E., & Yi, J. H. (2020). Risk assessment scheme for mobile applications based on tree boosting. IEEE Access, 8, 48503-48514.

Kissel, R. (2011) Glossary of Key Information Security Terms, in NISTIR 7298 National Institute of Standards and Technology, 2011, URL http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf, [Accessed on 10-May-2019].

Ko, K., Son, Y., Kim, S., & Lee, Y. (2017). DisCO: A distributed and concurrent offloading framework for mobile edge cloud computing. In the *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)* (pp. 763-766). IEEE.

Kovachev, D., Renzel, D., Klamma, R., Cao, Y., 2010. Mobile community cloud computing emerges and evolves, in Mobile Data Management (MDM), 2010. In: Proceedings of the Eleventh International Conference on, pp. 393–395.

Kulkarni, P., & Khanai, R. (2015). Addressing mobile Cloud Computing security issues: A survey. In *2015 International Conference on Communications and Signal Processing (ICCSP)* (pp. 1463-1467). IEEE.

Kulkarni, P., Khanai, R., & Bindagi, G. (2016). Security frameworks for mobile cloud computing: A survey. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)* (pp. 2507-2511). IEEE.

Kumar, M., & Hanumanthappa, M. (2015). Cloud-based intrusion detection architecture for smartphones. In *2015 international conference on innovations in information, embedded and communication systems (ICIIECS)* (pp. 1-6). IEEE.

Kumar, N., Singh, J. P., Bali, R. S., Misra, S., & Ullah, S. (2015). An intelligent clustering scheme for distributed intrusion detection in vehicular cloud computing. *Cluster Computing*, *18*(3), 1263-1283.

Kumar, R., & Goyal, R. (2019). On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, *33*, 1-48.

La Polla, M., Martinelli, F., & Sgandurra, D. (2012). A survey on security for mobile devices. *IEEE communications surveys & tutorials*, *15*(1), 446-471.

Lei, L., Sengupta, S., Pattanaik, T., & Gao, J. (2015). MCloudDB: A Mobile Cloud Database Service Framework. In 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (pp. 6-15). IEEE.

Liang, Y., Wang, W., Dong, K., Zhang, G., & Qi, G. (2021). Adoption of mobile government cloud from the perspective of public sector. Mobile Information Systems.

Li, H., Lan, C., Fu, X., Wang, C., Li, F., & Guo, H. (2020). A secure and lightweight fine-grained data sharing scheme for mobile cloud computing. Sensors, 20(17), 4720.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7), 3216-3225.

Li, J., Zhai, L., Zhang, X., & Quan, D. (2014). Research of android malware detection based on network traffic monitoring. In *2014 9th IEEE Conference on Industrial Electronics and Applications* (pp. 1739-1744). IEEE.

Li, R., Shen, C., He, H., Gu, X., Xu, Z., & Xu, C. Z. (2017). A lightweight secure data sharing scheme for mobile cloud computing. *IEEE Transactions on Cloud Computing*, *6*(2), 344-357.

Li, Y., Du, M., & Xu, J. (2018). A New Distributed Intrusion Detection Method Based on Immune Mobile Agent. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)* (pp. 215-219). IEEE

Lima, A., Rosa, L., Cruz, T., & Simões, P. (2020). A Security Monitoring Framework for Mobile Devices. Electronics, 9(8), 1197.

Lin, C., Shen, Z., Chen, Q., & Sheldon, F. T. (2017). A data integrity verification scheme in mobile cloud computing. *Journal of Network and Computer Applications*, *77*, 146-151.

Lin, X. (2011). Survey on cloud-based mobile security and a new framework for improvement. In *2011 IEEE International Conference on Information and Automation* (pp. 710-715). IEEE.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L.and Leaf, D. (2011) NIST Cloud Computing Reference Architecture (SP 500-292), National Institute of Standards & Technology, Gaithersburg, MD 20899-8930, USA, URL http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=909505,[Accessed on 10-May-2019].

Liu, L., Zhang, X., Yan, G., Chen, S., (2009). Exploitation and threat analysis of open mobile devices, In: Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 20–29.

Lordan, F., Jensen, J., & Badia, R. M. (2018). Towards mobile cloud computing with single sign-on access. Journal of Grid Computing, 16(4), 627-646.

Lu, H., Xia, X., & Wang, X. (2012). How to dynamically protect data in mobile cloud computing?. In Joint International Conference on Pervasive Computing and the Networked World (pp. 364-371). Springer, Berlin, Heidelberg.

Lu, N., Li, D., Shi, W., Vijayakumar, P., Piccialli, F., & Chang, V. (2021). An efficient combined deep neural network based malware detection framework in 5G environment. Computer Networks, 189, 107932.

Luo, B., & Xia, J. (2014). A novel intrusion detection system based on feature generation with visualization strategy. Expert Systems with Applications, 41(9), 4139-4147.

Mahdavifar, S., Kadir, A. F. A., Fatemi, R., Alhadidi, D., & Ghorbani, A. A. (2020). Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech) (pp. 515-522). IEEE.

Manikanthan, S. V., Padmapriya, T., Hussain, A., & Thamizharasi, E. (2020). Artificial Intelligence Techniques for Enhancing Smartphone Application Development on Mobile Computing, 4-19.

Man, N. D., & Huh, E. N. (2012). A collaborative intrusion detection system framework for cloud computing. In *Proceedings of the International Conference on IT Convergence and Security 2011* (pp. 91-109). Springer, Dordrecht.

Marengereke, T. M., & Sornalakshmi, K. (2015). Cloud-based security solution for Android smartphones. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (pp. 1-6). IEEE.

Marforio, C., Masti, R. J., Soriente, C., Kostiainen, K., & Capkun, S. (2016). Hardened setup of personalized security indicators to counter phishing attacks in mobile banking. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 83-92).

Marlinspike, M. (2009). More tricks for defeating SSL in practice. Black Hat USA, 516.

Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q., & Javaid, A. Y. (2021). NATICUSdroid: A malware detection framework for Android using native and custom permissions. Journal of Information Security and Applications, 58, 102696.

Mat, S. R. T., Ab Razak, M. F., Kahar, M. N. M., Arif, J. M., & Firdaus, A. (2022). A Bayesian probability model for Android malware detection. ICT Express, 8(3), 424-431.

Maza, S., & Touahria, M. (2019). Feature selection for intrusion detection using new multi-objective estimation of distribution algorithms. Applied Intelligence, 1-21.

Meads, A., Roughton, A., Warren, I., & Weerasinghe, T. (2009). Mobile service provisioning middleware for multihomed devices. In *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications* (pp. 67-72). IEEE.

Mehmood, Y., Shibli, M. A., Kanwal, A., & Masood, R. (2015). Distributed intrusion detection system using mobile agents in the cloud computing environment. In *2015 Conference on Information Assurance and Cyber Security (CIACS)* (pp. 1-8). IEEE.

Microsoft(2009) The STRIDE Threat Model, URL https://docs.microsoft.com/enus/previous-versions/commerce-server/ee823878(v=cs.20), [Accessed on 27-Oct-2019].

Microsoft (2022) "STRIDE Threat Modelling Tool" https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats

Milligan, P.M., Hutcheson, D., (2008). Business risks and security assessment for mobile devices. Inf. Syst. Control J. 1, 24.

Milosevic, J., Dittrich, A., Ferrante, A., & Malek, M. (2014). A resource-optimized approach to efficient early detection of mobile malware. In *2014 Ninth International Conference on Availability, Reliability and Security* (pp. 333-340). IEEE.

Modi, C. N. (2015). Network intrusion detection in cloud computing. In *Emerging Research in Computing, Information, Communication and Applications* (pp. 289-296). Springer, New Delhi.

Modi, C., Patel, D., Borisanya, B., Patel, A., & Rajarajan, M. (2012). A novel framework for intrusion detection in the cloud. In *Proceedings of the fifth international conference on security of information and networks* (pp. 67-74). ACM.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., & Rajarajan, M. (2013). A survey of intrusion detection techniques in the cloud. *Journal of network and computer applications*, *36*(1), 42-57.

Modi, C., & Patel, D. (2018). A feasible approach to intrusion detection in the virtual network layer of Cloud computing. *Sādhanā*, *43*(7), 114.

Mogull, R., Arlen, J., Lane, A., Peterson, K., Rothman, M., Mortman, D.(2017) Security Guidance for Critical Areas of Focus in Cloud Computing, Cloud Security Alliance, 2017, URL https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/security-guidance-v4-FINAL.pdf, [Accessed on 10-May-2019].

Mohammadi, S., Mirvaziri, H., Ghazizadeh-Ahsaee, M., & Karimipour, H. (2019). Cyber intrusion detection by combined feature selection algorithm. Journal of information security and applications, 44, 80-88.

Mohiuddin, K., Islam, A., Alam, A., and Ali, A. (2012): mobile cloud access. In Proceedings of the CUBE International Information Technology Conference, Pp: 544-551.

Mogal, D. G., Ghungrad, S. R., & Bhusare, B. B. (2017). NIDS using machine learning classifiers on UNSW-NB15 and KDDCUP99 datasets. International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), 6(4), 533-537.

Mollah, M. B., Azad, M. A. K., & Vasilakos, A. (2017). Security and privacy challenges in mobile cloud computing: Survey and way ahead. *Journal of Network and Computer Applications*, *84*, 38-54.

Mollah, M. B., Islam, K. R., & Islam, S. S. (2012, April). Next-generation computing through cloud computing technology. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1-6). IEEE.

Moloja, D., & Mpekoa, N. (2017). Towards a cloud intrusion detection and prevention system form-voting in south Africa. In *2017 International Conference on Information Society (i-Society)* (pp. 34-39). IEEE.

Moorthy, S. M., & Masillamani, M. R. (2014). Intrusion detection in cloud computing implementation of (SAAS & IAAS) using grid environment. In *Proceedings of International Conference on Internet Computing and Information Communications* (pp. 53-64). Springer, New Delhi.

Moorthy, V., Venkataraman, R., & Rao, T. R. (2020). Security and privacy attacks during data communication in software defined mobile clouds. Computer Communications, 153, 515-526.

Moustafa, N., & Slay, J. (2017). A hybrid feature selection for network intrusion detection systems: Central points. arXiv preprint arXiv:1707.05505.

Moustafa, N., Hu, J., & Slay, J. (2019). A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, *128*, 33-55.

Mugabo, E., & Zhang, Q. Y. (2020). Intrusion Detection Method Based on Support Vector Machine and Information Gain for Mobile Cloud Computing. International Journal. Network. Security, 22(2), 231-241.

Myasnikov, V., Shaposhnikov, A., Sartasov, S., Gordienko, E., Aphonina, O., & Gamaonov, A. (2021). Navitas Framework: A Novel Tool for Android Applications Energy Profiling. In Sixth Conference on Software Engineering and Information Management (SEIM-2021)(full papers) (p. 11).

Nagar, U., Nanda, P., He, X., & Tan, Z. T. (2017). A framework for data security in the cloud using a collaborative intrusion detection scheme. In *Proceedings of the 10th International Conference on Security of Information and Networks* (pp. 188-193). ACM.

Nezarat, A. (2017). A game-theoretic method for VM-to-hypervisor attacks detection in the cloud environment. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (pp. 1127-1132). IEEE.

Nezarat, A., & Shams, Y. (2017). A game theoretic-based distributed detection method for VM-to-hypervisor attacks in the cloud environment. *The Journal of Supercomputing*, *73*(10), 4407-4427.

Nguyen, D. C., Pathirana, P. N., Ding, M., & Seneviratne, A. (2019). Blockchain for secure ehrs sharing of mobile cloud based e-health systems. IEEE access, 7, 66792-66806.

Nguyen, K. K., Hoang, D. T., Niyato, D., Wang, P., Nguyen, D., & Dutkiewicz, E. (2018). Cyberattack detection in mobile cloud computing: A deep learning approach. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 1-6). IEEE.

Nisha, O. J., & Bhanu, S. M. S. (2020). Detection of malware applications using social spider algorithm in the mobile cloud computing environment. International Journal of Ad Hoc and Ubiquitous Computing, 34(3), 154-169.

Noor, T. H., Zeadally, S., Alfazi, A., & Sheng, Q. Z. (2018). Mobile cloud computing: Challenges and future research directions. *Journal of Network and Computer Applications*, *115*, 70-85.

Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009). Outline of a design science research process. In Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology (pp. 1-11).

Omri, F., Foufou, S., Hamila, R., & Jarraya, M. (2013). Cloud-based mobile system for biometrics authentication. In *2013 13th International Conference on ITS Telecommunications (ITST)* (pp. 325-330). IEEE.

Osanaiye, O., Cai, H., Choo, K.-K. R., Dehghantanha, A., Xu, Z., & Dlodlo, M. (2016). Ensemble based multi-filter feature selection method for DDoS detection in cloud computing. EURASIP Journal on Wireless Communications and Networking, 2016(1), 130.

OS, J. N. (2021). Detection of malicious Android applications using Ontology-based intelligent model in mobile cloud environment. Journal of Information Security and Applications, 58, 102751.

OWASP, (2021). Top 10 Most Critical Web Application Security Risks, Tech. Rep. Top 10 - 2021, The OWASP Foundation, URL https://owasp.org/www-project-top-ten/, [Accessed on 28-Jan-2022].

Panah, A., Panah, A., Panah, O., & Fallahpour, S. (2012). Challenges of security issues in cloud computing layers. Rep. Opin, 4(10), 25-29.

Pandeeswari, N., & Kumar, G. (2016). An anomaly detection system in a cloud environment using fuzzy clustering-based ANN. *Mobile Networks and Applications*, *21*(3), 494-505.

Pandian, V. A., & Kumar, T. G. (2014). A Novel Cloud-Based NIDPS for Smartphones. In *International Conference on Security in Computer Networks and Distributed Systems*(pp. 473-484). Springer, Berlin, Heidelberg.

Patel, A., Taghavi, M., Bakhtiyari, K., & Júnior, J. C. (2012). Taxonomy and proposed architecture of intrusion detection and prevention systems for cloud computing. In *Cyberspace Safety and Security* (pp. 441-458). Springer, Berlin, Heidelberg

Patel, A., Taghavi, M., Bakhtiyari, K., & JúNior, J. C. (2013). An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of network and computer applications*, *36*(1), 25-41.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems, 24*(3), 45-77.

Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., ... & Molloy, I. (2012). Using probabilistic generative models for ranking risks of android apps. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 241-252).

Peng, K., Zheng, L., Xu, X., Lin, T., & Leung, V. C. (2018). Balanced Iterative Reducing and Clustering Using Hierarchies with Principal Component Analysis (PBirch) for Intrusion Detection over Big Data in Mobile Cloud Environment. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage* (pp. 166-177). Springer, Cham.

Pham, N. T., Foo, E., Suriadi, S., Jeffrey, H., & Lahza, H. F. M. (2018). Improving performance of intrusion detection system using ensemble methods and feature selection. In Proceedings of the Australasian computer science week multiconference (pp. 1-6).

Pokharel, S., Choo, K.-K.R., Liu, J., (2017). Mobile cloud security: an adversary model for lightweight browser security. Computer. Stand. Interfaces 49, 71–78.

Pontarelli, S., Bianchi, G., & Teofili, S. (2012). Traffic-aware design of a high-speed FPGA network intrusion detection system. *IEEE Transactions on Computers*, *62*(11), 2322-2334.

Prandini, M., Ramilli, M., Cerroni, W., & Callegati, F. (2010). Splitting the HTTPS stream to attack secure web connections. IEEE Security & Privacy, 8(6), 80-84.

Prokhorenko, V., Choo, K.-K.R., Ashman, H., (2016). Web application protection techniques: a taxonomy. Journal. of Network and Computer Application. 60, 95–112.

Qin, T., Chen, R., Wang, L., & He, C. (2018). LMHADC: Lightweight Method for Host-based Anomaly Detection in Cloud using Mobile Agents. In *2018 IEEE Conference on Communications and Network Security (CNS)* (pp. 1-8). IEEE.

Qi, Y., Cao, M., Zhang, C., & Wu, R. (2014). A design of network behaviour-based malware detection system for Android. In *International Conference on Algorithms and Architectures for Parallel Processing* (pp. 590-600). Springer, Cham.

Quick, D., & Choo, K. K. R. (2017). Pervasive social networking forensics: intelligence and evidence from mobile device extracts. Journal of Network and Computer Applications, 86, 24-33.

Racic, R., Ma, D., Chen, H., (2006). Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery. In: Secure comm and Workshops, pp. 1–10.

Rai, P. O. (2013). Android Application Security Essentials. Packt Publishing Ltd.

Raja, S., & Ramaiah, S. (2017). An efficient fuzzy-based hybrid system to cloud intrusion detection. *International Journal of Fuzzy Systems*, *19*(1), 62-77.

Rajendran, R., Kumar, S. S., Palanichamy, Y., & Arputharaj, K. (2018) Detection of DoS attacks in cloud networks using the intelligent rule-based classification system. *Cluster Computing*, 1-12.

Rashidi, B., Fung, C., & Bertino, E. (2017). Android resource usage risk assessment using hidden Markov model and online learning. Computers & Security, 65, 90-107.

Ravji, S., & Ali, M. (2018). Integrated Intrusion Detection and Prevention System with Honeypot in Cloud Computing. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)* (pp. 95-100). IEEE.

Ren, W., Yu, L., Gao, R., & Xiong, F. (2011). Lightweight and compromise resilient storage outsourcing with distributed secure accessibility in mobile cloud computing. *Tsinghua Science and Technology*, *16*(5), 520-528.

Ribeiro, J., Mantas, G., Saghezchi, F. B., Rodriguez, J., Shepherd, S. J., & Abd-Alhameed, R. A. (2018). Towards an Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. In *International Conference on Broadband Communications, Networks and Systems* (pp. 139-148). Springer, Cham.

Ribeiro, J., Saghezchi, F. B., Mantas, G., Rodriguez, J., Shepherd, S. J., & Abd-Alhameed, R. A. (2019). An Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. *Mobile Networks and Applications*, 1-9.

Ribeiro, J., Saghezchi, F. B., Mantas, G., Rodriguez, J., Shepherd, S. J., & Abd-Alhameed, R. A. (2020). An autonomous host-based intrusion detection system for Android mobile devices. Mobile Networks and Applications, 25(1), 164-172.

Rittinghouse, J. W., & Ransome, J. F. (2017). Cloud computing: implementation, management, and security. CRC press.

Rodero-Merino, L., Vaquero, L. M., Caron, E., Muresan, A., & Desprez, F. (2012). Building safe PaaS clouds: A survey on security in multitenant software platforms. computers & security, 31(1), 96-108.

Roshandel, R., Arabshahi, P., & Poovendran, R. (2013). LIDAR: layered intrusion detection and remediation framework for smartphones. In *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems* (pp. 27-32). ACM.

Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. (2019). Analysis of software development methodologies. International Journal of Computing and Digital Systems, 8(5), 446-460.

Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2021). A novel Android malware detection system: adaption of filter-based feature selection methods. Journal of Ambient Intelligence and Humanized Computing, 1-15.

Sajjad, M., Muhammad, K., Baik, S. W., Rho, S., Jan, Z., Yeo, S. S., & Mehmood, I. (2017). Mobile-cloud assisted framework for selective encryption of medical images with steganography for resource-constrained devices. *Multimedia Tools and Applications*, *76*(3), 3519-3536.

Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2016). Madam: Effective and efficient behaviour-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing, 15(1), 83-97.

Sathye, M., Goundar, S., & Bhardwaj, A. (2022). Determinants of Mobile Cloud Computing Adoption by Financial Services Firms. Journal of Information Technology Research (JITR), 15(1), 1-17.

Sanaei, Z., Abolfazli, S., Gani, A., & Shiraz, M. (2012). SAMI: Service-based arbitrated multi-tier infrastructure for Mobile Cloud Computing. In *2012 1st IEEE International Conference on Communications in China Workshops (ICCC)*(pp. 14-19). IEEE.

Scandariato, R., Wuyts, K., & Joosen, W. (2015). A descriptive study of Microsoft's threat modelling technique. *Requirements Engineering, 20*(2), 163-180.

Scarfone, K., & Mell, P. (2007). Guide to intrusion detection and prevention systems (IDPS). *NIST special publication*, *800*(2007), 94.

Schaffer, H. E. (2009). X as a service, cloud computing, and the need for good judgment. *IT professional*, *11*(5), 4-5.

Sebastián, M., Rivera, R., Kotzias, P., & Caballero, J. (2016). Avclass: A tool for massive malware labeling. In *International symposium on research in attacks, intrusions, and defenses* (pp. 230-253). Springer, Cham.

Sgandurra, D., & Lupu, E. (2016). Evolution of attacks, threat models, and solutions for virtualized systems. ACM Computing Surveys (CSUR), 48(3), 1-38.

Shabbir, M., Shabbir, A., Iwendi, C., Javed, A. R., Rizwan, M., Herencsar, N., & Lin, J. C. W. (2021). Enhancing security of health information using modular encryption standard in mobile cloud computing. IEEE Access, 9, 8820-8834.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). "Andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 38(1), 161-190.

Sharma, K., & Gupta, B. B. (2018). Mitigation and risk factor analysis of android applications. Computers & Electrical Engineering, 71, 416-430.

Shi, Y., Abhilash, S., & Hwang, K. (2015). Cloudlet mesh for securing mobile clouds from intrusions and network attacks. In 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (pp. 109-118). IEEE

Shila, D. M., Shen, W., Cheng, Y., Tian, X., & Shen, X. S. (2016). AMCloud: Toward a secure autonomic mobile ad hoc cloud computing system. *IEEE Wireless Communications*, *24*(2), 74-81.

Shiny, R., Shaji, R. S., & Jayan, J. P. (2015). Signature-based data auditing under mobile cloud system. In *2015 Global Conference on Communication Technologies (GCCT)*(pp. 565-570). IEEE.

Singh, S., Jeong, Y. S., & Park, J. H. (2016). A survey on cloud computing security: Issues, threats, and solutions. Journal of Network and Computer Applications, 75, 200-222.

Singh, T., Verma, S., Kulshrestha, V., & Katiyar, S. (2016). Intrusion detection system using genetic algorithm for the cloud. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies* (p. 115). ACM.

Sohal, A. S., Sandhu, R., Sood, S. K., & Chang, V. (2018). A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments. *Computers & Security*, *74*, 340-354.

Son, H. X., Carminati, B., & Ferrari, E. (2021). A Risk Assessment Mechanism for Android Apps. In 2021 IEEE International Conference on Smart Internet of Things (SmartIoT) (pp. 237-244). IEEE.

Sood, S. K. (2012). A combined approach to ensure data security in cloud computing. Journal of Network and Computer Applications, 35(6), 1831-1838.

Statcounter(2022) https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/2022

Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, *34*(1), 1-11.

Subramaniam Govindaraj, S. P. (2020). Joint Honeypot Networks and Hybrid Intrusion Detection System for Mobile Cloud Computing (Doctoral dissertation, Dublin, National College of Ireland).

Sun, S., Ye, Z., Yan, L., Su, J., & Wang, R. (2018). Wrapper feature selection based on lightning attachment procedure optimization and support vector machine for intrusion detection. IEEE. Symposium conducted at the meeting of the 2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)

Sukumaran, S. C., & Misbahuddin, M. (2021). PCR and Bio-signature for data confidentiality and integrity in mobile cloud computing. Journal of King Saud University-Computer and Information Sciences, 33(4), 426-435.

Tama, B. A., Comuzzi, M., & Rhee, K.-H. (2019). TSE-IDS: A Two-Stage Classifier Ensemble for Intelligent Anomaly-Based Intrusion Detection System. IEEE Access, 7, 94497-94507.

Tahirkheli, A. I., Shiraz, M., Hayat, B., Idrees, M., Sajid, A., Ullah, R., ... & Kim, K. I. (2021). A survey on modern cloud computing security over smart city networks: Threats, vulnerabilities, consequences, countermeasures, and challenges. Electronics, 10(15), 1811.

Thumar, V., & Vekariya, V. (2016). A Framework for Secure Data Storage in Mobile Cloud Computing. In *Proceedings of International Conference on ICT for Sustainable Development* (pp. 791-800). Springer, Singapore.

Tong, F., & Yan, Z. (2017). A hybrid approach of mobile malware detection in Android. *Journal of Parallel and Distributed computing*, *103*, 22-31.

Tong, F., & Yan, Z. (2017). A hybrid approach of mobile malware detection in Android. Journal of Parallel and Distributed computing, 103, 22-31.

Toumi, H., Talea, M., Sabiri, K., & Eddaoui, A. (2015). Toward a trusted framework for cloud computing. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)* (pp. 1-6). IEEE.

Trabelsi, S., Di Cerbo, F., Gomez, L., & Bezzi, M. (2015). A Privacy-Preserving Framework for Mobile and Cloud. In *2015 2nd ACM International Conference on Mobile Software Engineering and Systems* (pp. 160-161). IEEE.

Ulltveit-Moe, N., Oleshchuk, V. A., & Køien, G. M. (2011). Location-aware mobile intrusion detection with enhanced privacy in a 5G context. *Wireless Personal Communications*, *57*(3), 317-338.

Velliangiri, S., & Premalatha, J. (2017). Intrusion detection of distributed denial of service attack in the cloud. *Cluster Computing*, 1-9.

Venable, J. R., Pries-Heje, J., & Baskerville, R. L. (2017). Choosing a design science research methodology. ACIS 2017 Proceedings. 112.

Vijayanand, R., Devaraj, D., & Kannapiran, B. (2018). Intrusion detection system for wireless mesh network using multiple support vector machine classifiers with genetic-algorithm-based feature selection. Computers & Security, 77, 304-314.

Wang, H., Si, J., Li, H., & Guo, Y. (2019). Rmvdroid: towards a reliable android malware dataset with app metadata. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (pp. 404-408). IEEE.

Wang, Y., Zheng, J., Sun, C., & Mukkamala, S. (2013). Quantitative security risk assessment of android permissions and applications. In IFIP Annual Conference on Data and Applications Security and Privacy (pp. 226-241). Springer, Berlin, Heidelberg.

Weng, Y., & Liu, L. (2019). A Collective Anomaly Detection Approach for Multidimensional Streams in Mobile Service Security. *IEEE Access*, *7*, 49157-49168.

Wu, J., Ding, L., Wu, Y., Min-Allah, N., Khan, S. U., & Wang, Y. (2014). C2detector: a covert channel detection framework in cloud computing. *Security and Communication Networks*, *7*(3), 544-557.

Xu, J., Yu, Y., Chen, Z., Cao, B., Dong, W., Guo, Y., & Cao, J. (2013). MobSafe: cloud computing-based forensic analysis for massive mobile applications using data mining. *Tsinghua science and technology*, *18*(4), 418-427.

Yang, X., Pan, T., & Shen, J. (2010). On 3G mobile e-commerce platform based on cloud computing. In 2010 3rd IEEE International Conference on Ubi-Media Computing (pp. 198-201). IEEE.

Yan, W. (2012). CAS: A framework of online detecting advanced malware families for cloud-based security. In *2012 1st IEEE International Conference on Communications in China (ICCC)* (pp. 220-225). IEEE.

Yassin, W., Udzir, N. I., Muda, Z., Abdullah, A., & Abdullah, M. T. (2012). A cloud-based intrusion detection service framework. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)* (pp. 213-218). IEEE.

Yazji, S., Scheuermann, P., Dick, R. P., Trajcevski, G., & Jin, R. (2014). Efficient location-aware intrusion detection to protect mobile devices. *Personal and Ubiquitous Computing*, *18*(1), 143-162.

Zhang, T., & Wen, F. (2016). An ID-Based Anonymous Authentication Scheme for Distributed Mobile Cloud Computing. In *International Conference on Geo-Informatics in Resource Management and Sustainable Ecosystem* (pp. 401-409). Springer, Singapore.

Zhang, J. Y., Wu, P., Zhu, J., Hu, H., & Bonomi, F. (2013). Privacy-preserved mobile sensing through a hybrid cloud trust framework. In *2013 IEEE Sixth International Conference on Cloud Computing* (pp. 952-953). IEEE.

Zhong, H., & Xiao, J. (2014). Design for a cloud-based hybrid Android application security assessment framework. In *2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS)* (pp. 539-546). IEEE.

Zhou, Q., Feng, F., Shen, Z., Zhou, R., Hsieh, M. Y., & Li, K. C. (2019). A novel approach for mobile malware classification and detection in Android systems. *Multimedia Tools and Applications*, *78*(3), 3529-3552.

Zhu, W., Luo, C., Wang, J., Li, S., (2011). Multimedia cloud computing. Signal Process.

Mag. IEEE 28, 59–69.

Zissis, D., & Lekkas, D. (2012). Addressing cloud computing security issues. *Future Generation computer systems*, *28*(3), 583-592.

Zkik, K., Orhanou, G., & El Hajji, S. (2017). Secure mobile multi-cloud architecture for authentication and data storage. *International Journal of Cloud Applications and Computing (IJCAC)*, *7*(2), 62-76.

APPENDIX A (TABLES)

Table 1 Unique List of Permissions Usage in the Constructed Dataset

S/N	Permission Name	Benign Apps	Malicious App	Total Usage
1	ACCESS_BACKGROUND_LOCATION	8	22	30
2	ACCESS_CHECKIN_PROPERTIES	1	20	21
3	ACCESS_COARSE_LOCATION	2465	12568	15033
4	ACCESS_FINE_LOCATION	2640	10970	13610
5	ACCESS_LOCATION_EXTRA_COMMANDS	137	4173	4310
6	ACCESS_MEDIA_LOCATION	1	0	1
7	ACCESS_NETWORK_STATE	9196	18037	27233
8	ACCESS_NOTIFICATION_POLICY	38	11	49
9	ACCESS_WIFI_STATE	3516	15361	18877
10	ACCOUNT_MANAGER	6	24	30
11	ACTIVITY_RECOGNITION	3	0	3
12	ADD_VOICEMAIL	0	1	1
13	ANSWER_PHONE_CALLS	9	2	11
14	BATTERY_STATS	53	301	354
15	BIND_ACCESSIBILITY_SERVICE	10	27	37
16	BIND_APPWIDGET	8	27	35
17	BIND_DEVICE_ADMIN	1	7	8
18	BIND_INPUT_METHOD	3	6	9
19	BIND_NOTIFICATION_LISTENER_SERVICE	8	5	13
20	BIND_PRINT_SERVICE	1	0	1
21	BIND_REMOTEVIEWS	3	0	3
22	BIND_SCREENING_SERVICE	1	0	1
23	BIND_TELECOM_CONNECTION_SERVICE	0	1	1
24	BIND_WALLPAPER	2	41	43
25	BLUETOOTH	676	1153	1829
26	BLUETOOTH_ADMIN	440	767	1207
27	BLUETOOTH_PRIVILEGED	14	4	18
28	BODY_SENSORS	7	28	35
29	BROADCAST_PACKAGE_REMOVED	1	4	5
30	BROADCAST_SMS	2	44	46
31	BROADCAST_STICKY	159	927	1086
32	BROADCAST_WAP_PUSH	2	12	14
33	CALL_PHONE	753	3466	4219
34	CALL_PRIVILEGED	4	22	26
35	CAMERA	1911	3630	5541
36	CAPTURE_AUDIO_OUTPUT	6	8	14
37	CHANGE_COMPONENT_ENABLED_STATE	4	30	34
38	CHANGE_CONFIGURATION	94	915	1009
39	CHANGE_NETWORK_STATE	247	2013	2260

S/N	Permission Name	Benign Apps	Malicious App	Total Usage
40	CHANGE_WIFI_MULTICAST_STATE	83	170	253
41	CHANGE_WIFI_STATE	486	5789	6275
42	CLEAR_APP_CACHE	58	187	245
43	CONTROL_LOCATION_UPDATES	0	19	19
44	DELETE_CACHE_FILES	2	62	64
45	DELETE_PACKAGES	5	108	113
46	DIAGNOSTIC	0	12	12
47	DISABLE_KEYGUARD	231	1519	1750
48	DUMP	0	5	5
49	EXPAND_STATUS_BAR	46	347	393
50	FACTORY_TEST	0	3	3
51	FOREGROUND_SERVICE	335	82	417
52	GET_ACCOUNTS	1918	2605	4523
53	GET_ACCOUNTS_PRIVILEGED	1	3	4
54	GET_PACKAGE_SIZE	49	278	327
55	GET_TASKS	641	9245	9886
56	GLOBAL_SEARCH	0	3	3
57	INSTALL_LOCATION_PROVIDER	0	2	2
58	INSTALL_PACKAGES	14	566	580
59	INSTALL_SHORTCUT	15	11	26
60	INSTANT_APP_FOREGROUND_SERVICE	0	1	1
61	INTERNET	9760	18406	28166
62	KILL_BACKGROUND_PROCESSES	118	2092	2210
63	LOCATION_HARDWARE	6	10	16
64	MANAGE_DOCUMENTS	53	36	89
65	MANAGE_OWN_CALLS	3	0	3
66	MASTER_CLEAR	0	10	10
67	MEDIA_CONTENT_CONTROL	41	25	66
68	MODIFY_AUDIO_SETTINGS	467	2454	2921
69	MODIFY_PHONE_STATE	19	289	308
70	MOUNT_FORMAT_FILESYSTEMS	1	56	57
71	MOUNT_UNMOUNT_FILESYSTEMS	150	5634	5784
72	NFC	113	151	264
73	NFC_TRANSACTION_EVENT	1	0	1
74	PACKAGE_USAGE_STATS	73	136	209
75	PERSISTENT_ACTIVITY	7	38	45
76	PROCESS_OUTGOING_CALLS	70	649	719
77	READ_CALENDAR	277	161	438
78	READ_CALL_LOG	77	164	241
79	READ_CONTACTS	743	1644	2387
80	READ_EXTERNAL_STORAGE	3021	6159	9180

S/N	Permission Name	Benign Apps	Malicious App	Total Usage
81	READ_INPUT_STATE	2	5	7
82	READ_LOGS	183	5634	5817
83	READ_PHONE_NUMBERS	8	8	16
84	READ_PHONE_STATE	2553	17785	20338
85	READ_SMS	163	1406	1569
86	READ_SYNC_SETTINGS	109	116	225
87	READ_SYNC_STATS	59	58	117
88	REBOOT	2	16	18
89	RECEIVE_BOOT_COMPLETED	2302	7042	9344
90	RECEIVE_MMS	18	206	224
91	RECEIVE_SMS	223	1195	1418
92	RECEIVE_WAP_PUSH	11	86	97
93	RECORD_AUDIO	821	3718	4539
94	REORDER_TASKS	39	361	400
95	REQUEST_COMPANION_RUN_IN_BACKGROUND	1	0	1
96	REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	1	0	1
97	REQUEST_DELETE_PACKAGES	13	4	17
98	REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	27	45	72
99	REQUEST_INSTALL_PACKAGES	107	460	567
100	RESTART_PACKAGES	86	3249	3335
101	SEND_SMS	205	3199	3404
102	SET_ALARM	11	3	14
103	SET_ALWAYS_FINISH	0	10	10
104	SET_ANIMATION_SCALE	1	6	7
105	SET_DEBUG_APP	14	151	165
106	SET_PREFERRED_APPLICATIONS	2	23	25
107	SET_PROCESS_LIMIT	0	3	3
108	SET_TIME	3	13	16
109	SET_TIME_ZONE	3	44	47
110	SET_WALLPAPER	586	1078	1664
111	SET_WALLPAPER_HINTS	226	116	342
112	SIGNAL_PERSISTENT_PROCESSES	0	19	19
113	STATUS_BAR	1	21	22
114	SYSTEM_ALERT_WINDOW	769	5431	6200
115	TRANSMIT_IR	7	24	31
116	UNINSTALL_SHORTCUT	4	1	5
117	UPDATE_DEVICE_STATS	8	79	87
118	USE_BIOMETRIC	24	2	26
119	USE_FINGERPRINT	179	45	224
120	USE_FULL_SCREEN_INTENT	1	1	2
121	USE_SIP	11	36	47

S/N	Permission Name	Benign Apps	Malicious App	Total Usage
122	VIBRATE	3363	9383	12746
123	WAKE_LOCK	5758	8322	14080
124	WRITE_APN_SETTINGS	9	746	755
125	WRITE_CALENDAR	281	176	457
126	WRITE_CALL_LOG	33	88	121
127	WRITE_CONTACTS	298	515	813
128	WRITE_EXTERNAL_STORAGE	6284	16855	23139
129	WRITE_GSERVICES	1	20	21
130	WRITE_SECURE_SETTINGS	20	231	251
131	WRITE_SETTINGS	563	3040	3603
132	WRITE_SYNC_SETTINGS	136	126	262

Table 2 Unique List of Intent Usage in the Constructed Dataset

S/N	Intent Name	Benign Apps	Malicious Apps	Total Usage
1	action_MAIN	9865	18168	28033
2	category_LAUNCHER	9858	18032	27890
3	category_DEFAULT	3510	7304	10814
4	action_BOOT_COMPLETED	2347	5577	7924
5	action_PACKAGE_ADDED	384	5345	5729
6	action_VIEW	2474	3074	5548
7	category_BROWSABLE	2245	2614	4859
8	action_USER_PRESENT	275	3664	3939
9	action_PACKAGE_REMOVED	196	2607	2803
10	category_HOME	126	2041	2167
11	action_SEARCH	460	412	872
12	action_CREATE_SHORTCUT	62	779	841
13	action_MY_PACKAGE_REPLACED	650	29	679
14	action_SEND	402	231	633
15	action_PACKAGE_REPLACED	205	414	619
16	action_MEDIA_MOUNTED	62	424	486
17	category_LEANBACK_LAUNCHER	378	66	444
18	action_NEW_OUTGOING_CALL	68	373	441
19	action_MEDIA_BUTTON	317	82	399
20	action_PACKAGE_INSTALL	79	316	395
21	action_SCREEN_ON	26	266	292
22	category_MONKEY	12	255	267
23	action_TIMEZONE_CHANGED	156	109	265
24	action_SCREEN_OFF	27	214	241
25	category_INFO	86	150	236
26	action_MEDIA_EJECT	27	190	217
27	action_MEDIA_UNMOUNTED	37	177	214
28	action_BATTERY_CHANGED	22	150	172
29	action_BATTERY_LOW	126	45	171
30	action_BATTERY_OKAY	122	36	158
31	action_MEDIA_REMOVED	28	130	158
32	action_EDIT	52	100	152
33	action_SEND_MULTIPLE	95	52	147
34	action_PACKAGE_CHANGED	34	97	131
35	action_GET_CONTENT	60	69	129
36	action_SENDTO	34	94	128
37	action_LOCALE_CHANGED	82	42	124
38	action_DATE_CHANGED	29	94	123
39	category_OPENABLE	49	72	121

S/N	Intent Name	Benign Apps	Malicious Apps	Total Usage
40	category_PREFERENCE	12	104	116
41	action_DEVICE_STORAGE_LOW	95	14	109
42	action_HEADSET_PLUG	71	36	107
43	action_SET_WALLPAPER	57	47	104
44	action_DEVICE_STORAGE_OK	88	13	101
45	action_MEDIA_CHECKING	5	92	97
46	action_PACKAGE_RESTARTED	14	80	94
47	action_PICK	42	52	94
48	action_WALLPAPER_CHANGED	11	82	93
49	action_TIME_TICK	8	84	92
50	action_DIAL	18	52	70
51	action_MEDIA_BAD_REMOVAL	19	49	68
52	action_MEDIA_SCANNER_FINISHED	19	43	62
53	action_INSERT	8	52	60
54	action_CALL_BUTTON	5	52	57
55	action_MEDIA_SCANNER_STARTED	16	38	54
56	action_PACKAGE_DATA_CLEARED	21	29	50
57	action_REBOOT	25	25	50
58	action_MEDIA_SHARED	19	23	42
59	action_WEB_SEARCH	13	28	41
60	category_ALTERNATIVE	19	22	41
61	action_CALL	9	27	36
62	action_UMS_CONNECTED	2	34	36
63	action_MEDIA_NOFS	1	34	35
64	action_DEFAULT	2	29	31
65	action_PACKAGE_FULLY_REMOVED	25	6	31
66	action_INPUT_METHOD_CHANGED	2	26	28
67	action_EXTERNAL_APPLICATIONS_AVAILABLE	8	19	27
68	action_CONFIGURATION_CHANGED	6	20	26
69	action_DELETE	1	25	26
70	action_LOCKED_BOOT_COMPLETED	24	2	26
71	category_SAMPLE_CODE	3	20	23
72	category_APP_BROWSER	9	13	22
73	action_SEARCH_LONG_PRESS	2	17	19
74	action_PROCESS_TEXT	16	2	18
75	category_SELECTED_ALTERNATIVE	11	7	18
76	category_TAB	5	13	18
77	action_UMS_DISCONNECTED	0	17	17
78	action_CLOSE_SYSTEM_DIALOGS	2	14	16
79	action_MEDIA_SCANNER_SCAN_FILE	5	11	16

S/N	Intent Name	Benign Apps	Malicious Apps	Total Usage
80	action_MEDIA_UNMOUNTABLE	3	13	16
81	action_ASSIST	7	7	14
82	action_CAMERA_BUTTON	2	11	13
83	action_PROVIDER_CHANGED	7	5	12
84	action_EXTERNAL_APPLICATIONS_UNAVAILABLE	1	10	11
85	category_APP_MARKET	2	9	11
86	action_MANAGE_NETWORK_USAGE	9	1	10
87	action_PACKAGE_FIRST_LAUNCH	5	5	10
88	action_INSERT_OR_EDIT	1	8	9
89	category_DESK_DOCK	1	8	9
90	category_EMBED	4	5	9
91	action_APPLICATION_PREFERENCES	8	0	8
92	action_ATTACH_DATA	6	2	8
93	action_CHOOSER	4	3	7
94	action_DOCK_EVENT	0	7	7
95	action_RUN	2	5	7
96	action_UID_REMOVED	0	7	7
97	category_APP_MUSIC	7	0	7
98	action_ANSWER	3	3	6
99	action_DREAMING_STOPPED	1	5	6
100	action_USER_INITIALIZE	4	2	6
101	action_INSTALL_PACKAGE	1	4	5
102	action_PACKAGE_NEEDS_VERIFICATION	1	3	4
103	action_VOICE_COMMAND	2	2	4
104	category_APP_CALENDAR	4	0	4
105	category_CAR_DOCK	2	2	4
106	action_ALL_APPS	0	3	3
107	action_DREAMING_STARTED	1	2	3
108	action_MANAGE_PACKAGE_STORAGE	0	3	3
109	action_POWER_CONNECTED	2	1	3
110	action_TIME_CHANGED	0	3	3
111	action_UNINSTALL_PACKAGE	0	3	3
112	category_APP_CONTACTS	2	1	3
113	category_APP_MESSAGING	2	1	3
114	category_CAR_MODE	1	2	3
115	category_VOICE	3	0	3
116	action_APPLICATION_RESTRICTIONS_CHANGED	0	2	2
117	action_GET_RESTRICTION_ENTRIES	0	2	2
118	action_SYNC	2	0	2
119	category_APP_GALLERY	1	1	2

S/N	Intent Name	Benign Apps	Malicious Apps	Total Usage
120	category_DEVELOPMENT_PREFERENCE	1	1	2
121	action_BUG_REPORT	1	0	1
122	action_CREATE_DOCUMENT	1	0	1
123	action_OPEN_DOCUMENT	1	0	1
124	action_OPEN_DOCUMENT_TREE	1	0	1
125	action_PACKAGE_VERIFIED	1	0	1
126	action_POWER_DISCONNECTED	1	0	1
127	action_SHOW_APP_INFO	1	0	1
128	action_USER_UNLOCKED	0	1	1
129	category_APP_EMAIL	1	0	1
130	category_TEST	1	0	1
131	action_AIRPLANE_MODE_CHANGED	0	1	1

Table 3 Feature Selection Results of the Proposed Filter-Based FS Method

ID	B App%	M App%	δ	ε	Status	ID	B App%	M App%	δ	ε	Status
PI1	0.08	0.12	0.04	0.68	VOID	PI54	0.50	1.51	1.01	0.33	VOID
PI2	0.01	0.11	0.10	0.09	VOID	PI55	6.49	50.17	43.68	0.13	GOOD
PI3	24.95	68.20	43.25	0.37	GOOD	PI56	0.00	0.02	0.02	0.00	VOID
PI4	26.72	59.53	32.81	0.45	GOOD	PI57	0.00	0.01	0.01	0.00	VOID
PI5	1.39	22.65	21.26	0.06	GOOD	PI58	0.14	3.07	2.93	0.05	VOID
PI6	0.01	0.00	0.01	0.00	VOID	PI59	0.15	0.06	0.09	0.39	VOID
PI7	93.09	97.88	4.80	0.95	VOID	PI60	0.00	0.01	0.01	0.00	VOID
PI8	0.38	0.06	0.32	0.16	VOID	PI61	98.80	99.89	1.09	0.99	VOID
PI9	35.59	83.36	47.77	0.43	GOOD	PI62	1.19	11.35	10.16	0.11	GOOD
PI10	0.06	0.13	0.07	0.47	VOID	PI63	0.06	0.05	0.01	0.89	VOID
PI11	0.03	0.00	0.03	0.00	VOID	PI64	0.54	0.20	0.34	0.36	VOID
PI12	0.00	0.01	0.01	0.00	VOID	PI65	0.03	0.00	0.03	0.00	VOID
PI13	0.09	0.01	0.08	0.12	VOID	PI66	0.00	0.05	0.05	0.00	VOID
PI14	0.54	1.63	1.10	0.33	VOID	PI67	0.42	0.14	0.28	0.33	VOID
PI15	0.10	0.15	0.05	0.69	VOID	PI68	4.73	13.32	8.59	0.35	GOOD
PI16	0.08	0.15	0.07	0.55	VOID	PI69	0.19	1.57	1.38	0.12	VOID
PI17	0.01	0.04	0.03	0.27	VOID	PI70	0.01	0.30	0.29	0.03	VOID
PI18	0.03	0.03	0.00	0.93	VOID	PI71	1.52	30.57	29.06	0.05	GOOD
PI19	0.08	0.03	0.05	0.34	VOID	PI72	1.14	0.82	0.32	0.72	VOID
PI20	0.01	0.00	0.01	0.00	VOID	PI73	0.01	0.00	0.01	0.00	VOID
PI21	0.03	0.00	0.03	0.00	VOID	PI74	0.74	0.74	0.00	1.00	VOID
PI22	0.01	0.00	0.01	0.00	VOID	PI75	0.07	0.21	0.14	0.34	VOID
PI23	0.00	0.01	0.01	0.00	VOID	PI76	0.71	3.52	2.81	0.20	VOID
PI24	0.02	0.22	0.20	0.09	VOID	PI77	2.80	0.87	1.93	0.31	VOID
PI25	6.84	6.26	0.59	0.91	VOID	PI78	0.78	0.89	0.11	0.88	VOID
PI26	4.45	4.16	0.29	0.93	VOID	PI79	7.52	8.92	1.40	0.84	GOOD
PI27	0.14	0.02	0.12	0.15	VOID	PI80	30.58	33.42	2.84	0.91	GOOD
PI28	0.07	0.15	0.08	0.47	VOID	PI81	0.02	0.03	0.01	0.75	VOID
PI29	0.01	0.02	0.01	0.47	VOID	PI82	1.85	30.57	28.72	0.06	GOOD
PI30	0.02	0.24	0.22	0.08	VOID	PI83	0.08	0.04	0.04	0.54	VOID
PI31	1.61	5.03	3.42	0.32	GOOD	PI84	25.84	96.52	70.67	0.27	GOOD
PI32	0.02	0.07	0.04	0.31	VOID	PI85	1.65	7.63	5.98	0.22	GOOD
PI33	7.62	18.81	11.19	0.41	GOOD	PI86	1.10	0.63	0.47	0.57	VOID
PI34	0.04	0.12	0.08	0.34	VOID	PI87	0.60	0.31	0.28	0.53	VOID
PI35	19.34	19.70	0.36	0.98	VOID	PI88	0.02	0.09	0.07	0.23	VOID
PI36	0.06	0.04	0.02	0.71	VOID	PI89	23.30	38.22	14.91	0.61	GOOD
PI37	0.04	0.16	0.12	0.25	VOID	PI90	0.18	1.12	0.94	0.16	VOID
PI38	0.95	4.97	4.01	0.19	GOOD	PI91	2.26	6.49	4.23	0.35	GOOD
PI39	2.50	10.92	8.42	0.23	GOOD	PI92	0.11	0.47	0.36	0.24	VOID
PI40	0.84	0.92	0.08	0.91	VOID	PI93	8.31	20.18	11.87	0.41	GOOD
PI41	4.92	31.42	26.50	0.16	GOOD	PI94	0.39	1.96	1.56	0.20	VOID
PI42	0.59	1.01	0.43	0.58	VOID	PI95	0.01	0.00	0.01	0.00	VOID
PI43	0.00	0.10	0.10	0.00	VOID	PI96	0.01	0.00	0.01	0.00	VOID
PI44	0.02	0.34	0.32	0.06	VOID	PI97	0.13	0.02	0.11	0.16	VOID
PI45	0.05	0.59	0.54	0.09	VOID	PI98	0.27	0.24	0.03	0.89	VOID
PI46	0.00	0.07	0.07	0.00	VOID	PI99	1.08	2.50	1.41	0.43	VOID
PI47	2.34	8.24	5.91	0.28	GOOD	PI100	0.87	17.63	16.76	0.05	GOOD
PI48	0.00	0.03	0.03	0.00	VOID	PI101	2.08	17.36	15.29	0.12	GOOD
PI49	0.47	1.88	1.42	0.25	VOID	PI102	0.11	0.02	0.10	0.15	VOID
PI50	0.00	0.02	0.02	0.00	VOID	PI103	0.00	0.05	0.05	0.00	VOID
PI51	3.39	0.44	2.95	0.13	VOID	PI104	0.01	0.03	0.02	0.31	VOID
PI52	19.41	14.14	5.28	0.73	GOOD	PI105	0.14	0.82	0.68	0.17	VOID
PI53	0.01	0.02	0.01	0.62	VOID	PI106	0.02	0.12	0.10	0.16	VOID

ID	B App%	M App%	δ	ε	Status	ID	B App%	M App%	δ	ε	Status
PI107	0.00	0.02	0.02	0.00	VOID	PI162	0.53	0.54	0.02	0.97	VOID
PI108	0.03	0.07	0.04	0.43	VOID	PI163	0.08	0.10	0.02	0.79	VOID
PI109	0.03	0.24	0.21	0.13	VOID	PI164	0.01	0.05	0.04	0.19	VOID
PI110	5.93	5.85	0.08	0.99	VOID	PI165	0.61	0.37	0.23	0.62	VOID
PI111	2.29	0.63	1.66	0.28	VOID	PI166	0.00	0.01	0.01	0.00	VOID
PI112	0.00	0.10	0.10	0.00	VOID	PI167	0.72	0.20	0.52	0.27	VOID
PI113	0.01	0.11	0.10	0.09	VOID	PI168	0.02	0.14	0.12	0.14	VOID
PI114	7.78	29.47	21.69	0.26	GOOD	PI169	0.08	0.28	0.20	0.29	VOID
PI115	0.07	0.13	0.06	0.54	VOID	PI170	0.01	0.04	0.03	0.23	VOID
PI116	0.04	0.01	0.04	0.13	VOID	PI171	0.01	0.02	0.01	0.47	VOID
PI117	0.08	0.43	0.35	0.19	VOID	PI171	0.83	0.23	0.60	0.27	VOID
PI118	0.24	0.01	0.23	0.04	VOID	PI172	0.24	0.01	0.23	0.04	VOID
PI119	1.81	0.24	1.57	0.13	VOID	PI173	99.86	98.59	1.26	0.99	VOID
PI120	0.01	0.01	0.00	0.54	VOID	PI174	0.09	0.01	0.09	0.06	VOID
PI121	0.11	0.20	0.08	0.57	VOID	PI175	0.00	0.02	0.02	0.00	VOID
PI122	34.04	50.92	16.88	0.67	GOOD	PI176	0.19	0.27	0.07	0.72	VOID
PI123	58.29	45.16	13.12	0.77	GOOD	PI177	3.21	0.44	2.76	0.14	VOID
PI124	0.09	4.05	3.96	0.02	VOID	PI178	0.05	0.50	0.45	0.10	VOID
PI125	2.84	0.96	1.89	0.34	VOID	PI179	0.27	1.03	0.76	0.27	VOID
PI126	0.33	0.48	0.14	0.70	VOID	PI180	0.63	2.30	1.67	0.27	VOID
PI127	3.02	2.79	0.22	0.93	VOID	PI181	0.01	0.18	0.17	0.05	VOID
PI128	63.61	91.47	27.86	0.70	GOOD	PI182	0.28	0.71	0.42	0.40	VOID
PI129	0.01	0.11	0.10	0.09	VOID	PI183	0.19	0.23	0.04	0.82	VOID
PI130	0.20	1.25	1.05	0.16	VOID	PI184	0.05	0.06	0.01	0.85	VOID
PI131	5.70	16.50	10.80	0.35	GOOD	PI185	0.16	0.21	0.04	0.79	VOID
PI132	1.38	0.68	0.69	0.50	VOID	PI186	0.19	0.12	0.07	0.65	VOID
PI133	0.00	0.00	0.00	0.00	VOID	PI187	0.03	0.07	0.04	0.43	VOID
PI134	0.00	0.02	0.02	0.00	VOID	PI188	0.37	0.96	0.59	0.39	VOID
PI135	0.03	0.02	0.01	0.54	VOID	PI190	6.58	0.16	6.42	0.02	VOID
PI136	0.08	0.00	0.08	0.00	VOID	PI191	0.69	2.02	1.34	0.34	VOID
PI137	0.00	0.01	0.01	0.00	VOID	PI192	0.01	0.00	0.01	0.00	VOID
PI138	0.07	0.04	0.03	0.54	VOID	PI193	0.01	0.00	0.01	0.00	VOID
PI139	0.06	0.01	0.05	0.18	VOID	PI194	3.89	29.01	25.12	0.13	GOOD
PI140	0.22	0.81	0.59	0.27	VOID	PI195	0.34	0.53	0.18	0.65	VOID
PI141	1.28	0.24	1.03	0.19	VOID	PI196	0.21	0.16	0.06	0.74	VOID
PI142	1.23	0.20	1.04	0.16	VOID	PI197	0.05	0.03	0.02	0.54	VOID
PI143	23.76	30.27	6.51	0.78	GOOD	PI198	0.25	0.03	0.22	0.13	VOID
PI144	0.01	0.00	0.01	0.00	VOID	PI199	0.80	1.71	0.92	0.47	VOID
PI145	0.09	0.15	0.06	0.62	VOID	PI200	0.01	0.02	0.01	0.62	VOID
PI146	0.05	0.28	0.23	0.18	VOID	PI201	1.98	14.15	12.16	0.14	GOOD
PI147	0.02	0.06	0.04	0.34	VOID	PI202	2.08	2.25	0.17	0.92	VOID
PI148	0.04	0.02	0.02	0.40	VOID	PI203	0.14	0.43	0.29	0.33	VOID
PI149	0.02	0.08	0.06	0.27	VOID	PI204	0.01	0.00	0.01	0.00	VOID
PI150	0.06	0.11	0.05	0.56	VOID	PI205	0.43	0.28	0.14	0.66	VOID
PI151	0.01	0.00	0.01	0.00	VOID	PI206	0.02	0.01	0.01	0.27	VOID
PI152	0.63	4.23	3.60	0.15	VOID	PI207	0.01	0.00	0.01	0.00	VOID
PI153	0.29	0.51	0.22	0.58	VOID	PI208	0.16	0.01	0.15	0.07	VOID
PI154	0.02	0.16	0.14	0.13	VOID	PI209	0.07	0.03	0.04	0.38	VOID
PI155	0.01	0.14	0.13	0.07	VOID	PI210	0.25	0.14	0.12	0.54	VOID
PI156	0.96	0.08	0.89	0.08	VOID	PI211	0.02	0.03	0.01	0.75	VOID
PI157	0.89	0.07	0.82	0.08	VOID	PI212	0.27	1.16	0.89	0.24	VOID
PI158	0.18	0.28	0.10	0.65	VOID	PI213	0.26	1.44	1.18	0.18	VOID
PI159	0.00	0.04	0.04	0.00	VOID	PI214	4.66	2.24	2.42	0.48	GOOD
PI160	0.01	0.01	0.00	0.93	VOID	PI215	0.02	0.09	0.07	0.22	VOID
PI161	0.01	0.03	0.02	0.37	VOID	PI216	4.07	1.25	2.82	0.31	VOID
PI217	0.96	0.28	0.68	0.29	VOID	PI241	0.01	0.01	0.00	0.54	VOID

ID	B App%	M App%	δ	ε	Status	ID	B App%	M App%	δ	ε	Status
PI218	0.34	0.51	0.17	0.67	VOID	PI242	0.02	0.05	0.03	0.41	VOID
PI219	0.58	0.26	0.32	0.44	VOID	PI243	0.02	0.01	0.01	0.27	VOID
PI220	0.01	0.00	0.01	0.00	VOID	PI244	0.07	0.00	0.07	0.00	VOID
PI221	0.02	0.00	0.02	0.00	VOID	PI245	22.72	14.19	8.54	0.62	GOOD
PI222	0.00	0.02	0.02	0.00	VOID	PI246	0.02	0.01	0.01	0.54	VOID
PI223	0.08	0.46	0.37	0.18	VOID	PI247	0.01	0.01	0.00	0.93	VOID
PI224	1.58	0.59	0.99	0.37	VOID	PI248	35.53	39.64	4.11	0.90	GOOD
PI225	0.00	0.04	0.04	0.00	VOID	PI249	0.01	0.04	0.03	0.23	VOID
PI226	0.02	0.18	0.16	0.11	VOID	PI250	0.01	0.01	0.00	0.54	VOID
PI227	0.00	0.09	0.09	0.00	VOID	PI251	0.04	0.03	0.01	0.67	VOID
PI228	0.00	0.02	0.02	0.00	VOID	PI252	1.28	11.08	9.80	0.12	GOOD
PI229	0.04	0.01	0.03	0.27	VOID	PI253	0.87	0.81	0.06	0.94	VOID
PI230	2.78	19.88	17.10	0.14	GOOD	PI254	99.79	97.86	1.93	0.98	VOID
PI231	0.00	0.01	0.01	0.00	VOID	PI255	3.83	0.36	3.47	0.09	VOID
PI232	25.04	16.68	8.36	0.67	GOOD	PI256	0.12	1.38	1.26	0.09	VOID
PI233	0.02	0.01	0.01	0.54	VOID	PI257	0.50	0.39	0.11	0.79	VOID
PI234	0.11	0.44	0.33	0.25	VOID	PI258	0.12	0.56	0.44	0.22	VOID
PI235	0.13	0.15	0.02	0.87	VOID	PI259	0.03	0.11	0.08	0.28	VOID
PI236	0.19	0.12	0.07	0.62	VOID	PI260	0.11	0.04	0.07	0.34	VOID
PI237	0.09	0.07	0.02	0.77	VOID	PI261	0.05	0.07	0.02	0.72	VOID
PI238	0.04	0.00	0.04	0.00	VOID	PI262	0.01	0.00	0.01	0.00	VOID
PI239	0.02	0.01	0.01	0.27	VOID	PI263	0.03	0.00	0.03	0.00	VOID
PI240	0.01	0.00	0.01	0.00	VOID						

Note: PI1-PI132 represents the unique permissions list in Table 1 and PI133-PI263 represents the unique intents list in Table 2.

Table 4 Related Works and their ML Classifiers Used

Source	ML Classifier
Luo & Xia	SVM
Eesa et al.	DT
Bhattacharya & Selvakumar	BN, J48, KM
Osanaiye et al.	DT
Ambusaidi et al.	SVM
Belouch et al	REPTree
Moustafa & Slay	EM, LR, NB
Bostani & Sheikhan	SVM
Mogal et al.	NB, LR
Idhammad et al.	ANN
Vijayanand et al	SVM
Aljawarneh et al.	NB, J48, RT
Anwer et al.	J48, NB
Sun et al.	SVM
Pham et al.	EC, J48
Besharati et al.	EC, DT, LDA, ANN
Mohammadi et al.	DT
Çavuşoğlu	RF, J48, KNN, NB
Maza & Touahria	NB, MLP, SVM, KNN and DT
Tama et al.	RF, CF
Pandian & Kumar	BN, NB, SMO, J48 ,RF,RT DT
Qi et al.	NB
Modi	AC
Pandeeswari & Kumar	ANN
Raja & Ramaiah	Type-2 fuzzy TSK-rule
Velliangiri & Premalatha	Radial basis function neural network
Besharati et al.	ANN, DT, LDA
Kim et al.	K-means and DBSCAN method
Modi & Patel	BN, AC, DT
Rajendran et al.	rule based
Ribeiro et al.	OneR, DT, NB, BN, LR, SVM, k-NN

Note: DT- Decision Tree; SVM-Support Vector Machine; REPTree-Reduced Error Pruning Tree algorithm; EM-Expectation-Maximisation Clustering; LR-Logistic Regression; BN- Bayesian Networks; NB- Naïve Bayes; KM -K Means Learning Algorithm; ANN- Artificial Neural Networks; J48-J48 Decision Tree; EC-Ensemble Classifiers, Bagging or Boosting; CF- Conjunctive Rule; LDA-Linear Discriminant Analysis; MLP-Multilayer Perceptron; RF-Random Forest; SMO (Sequential Minimal Optimization);RT (Random Tree);AC-Associative classifier; LDA-linear discriminate analysis; OneR-One Rule

APPENDIX B (UML DIAGRAMS)

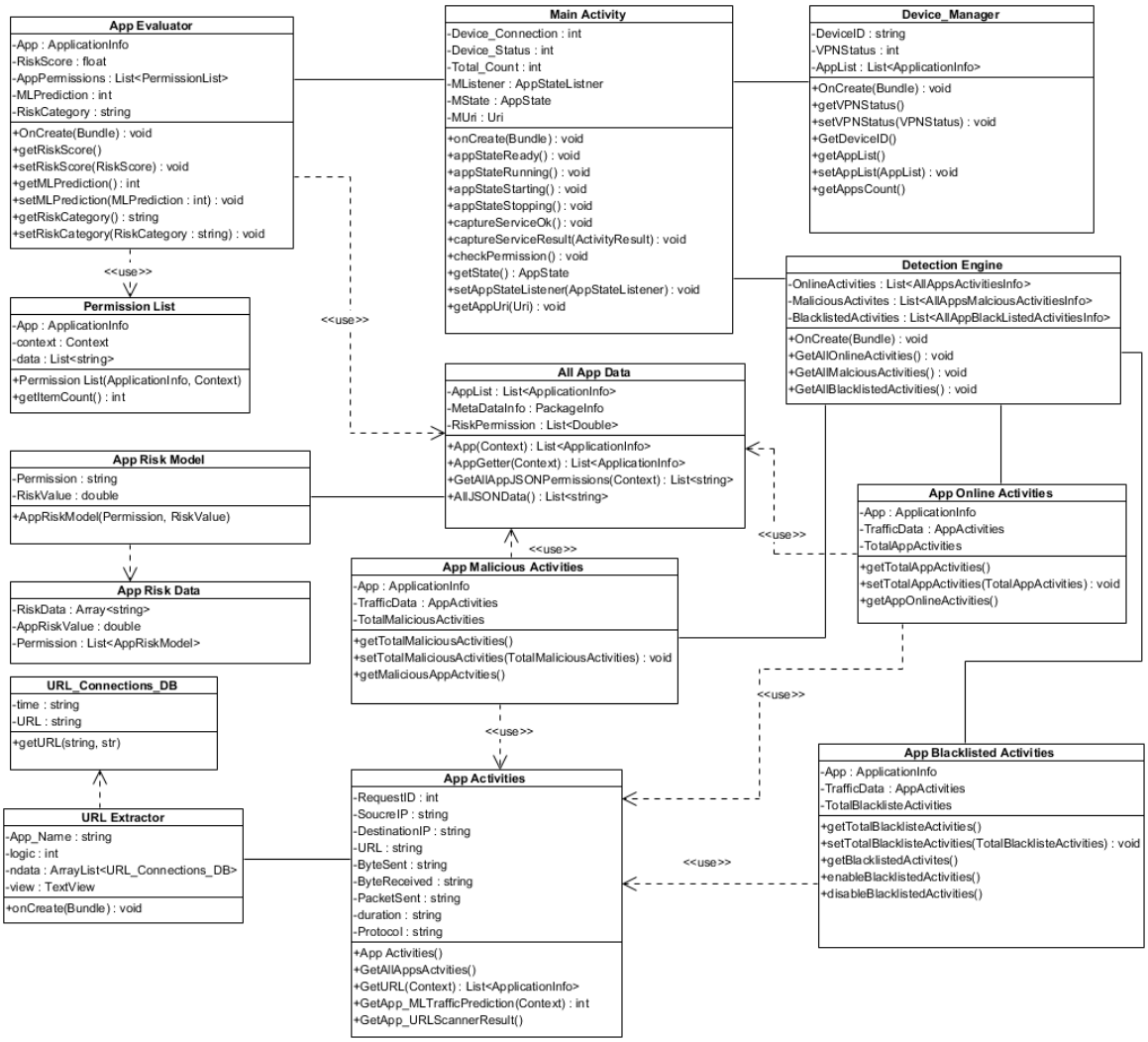

Figure 1. Class Diagram of the Prototype System

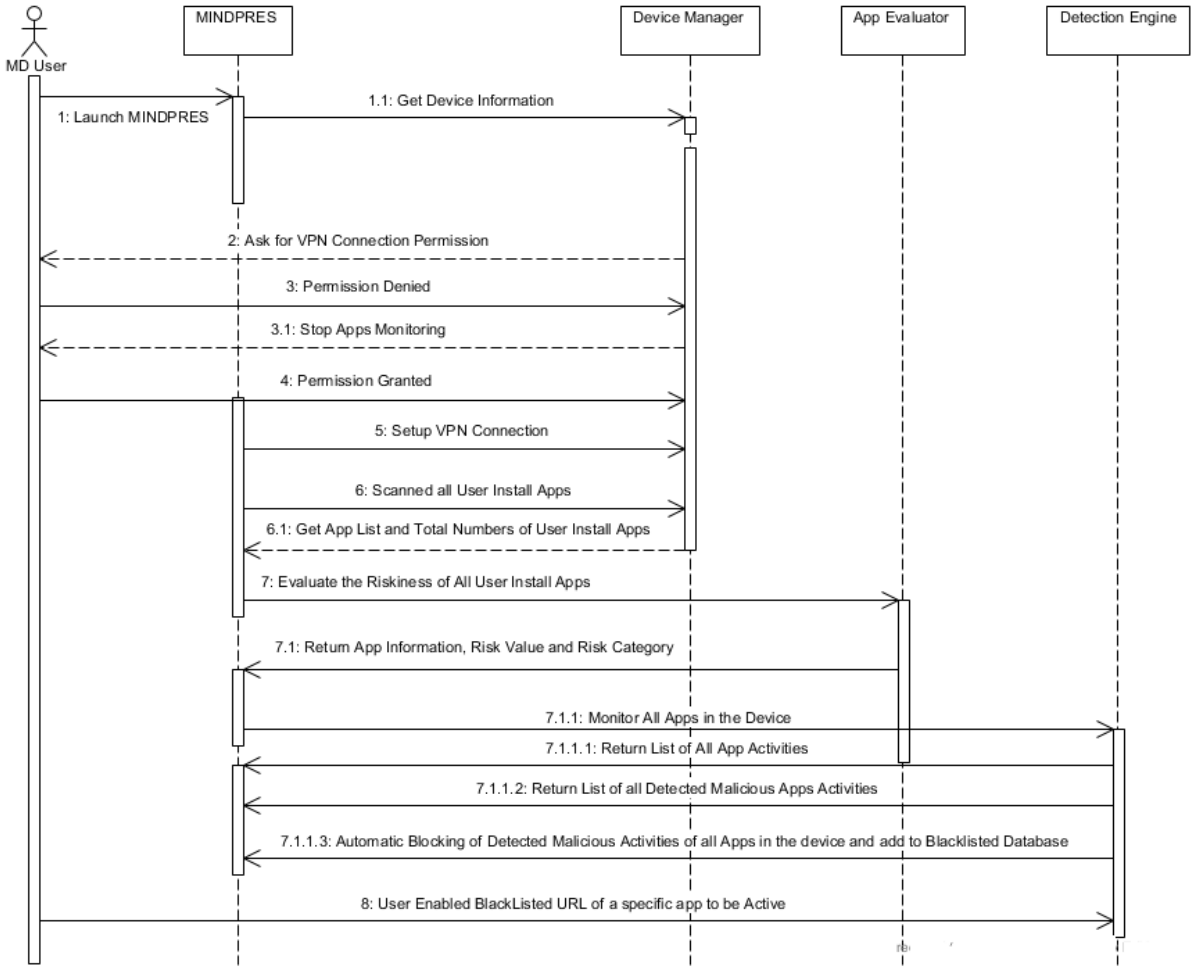

Figure 2. Sequence Diagram of the Prototype System

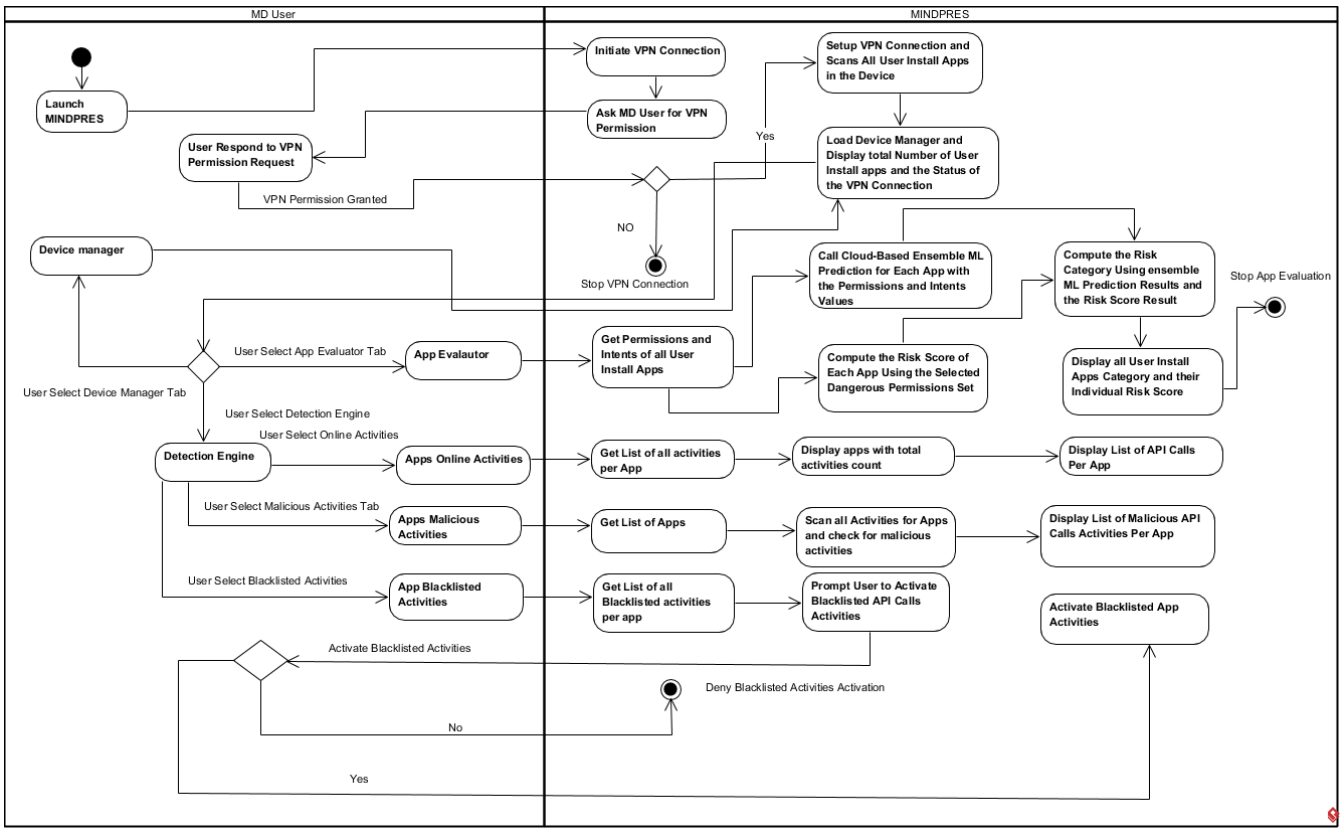

Figure 3. Activity Diagram of the Prototype System

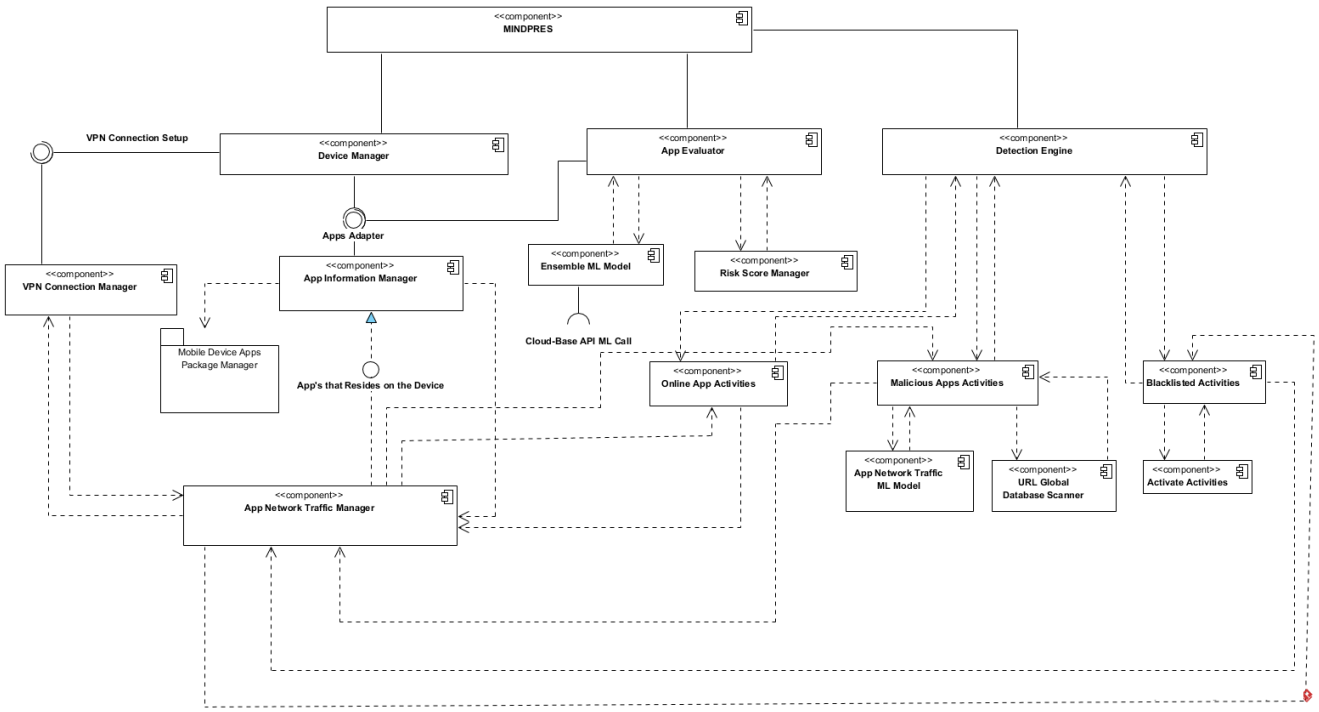

Figure 4. Component Diagram of the Prototype System

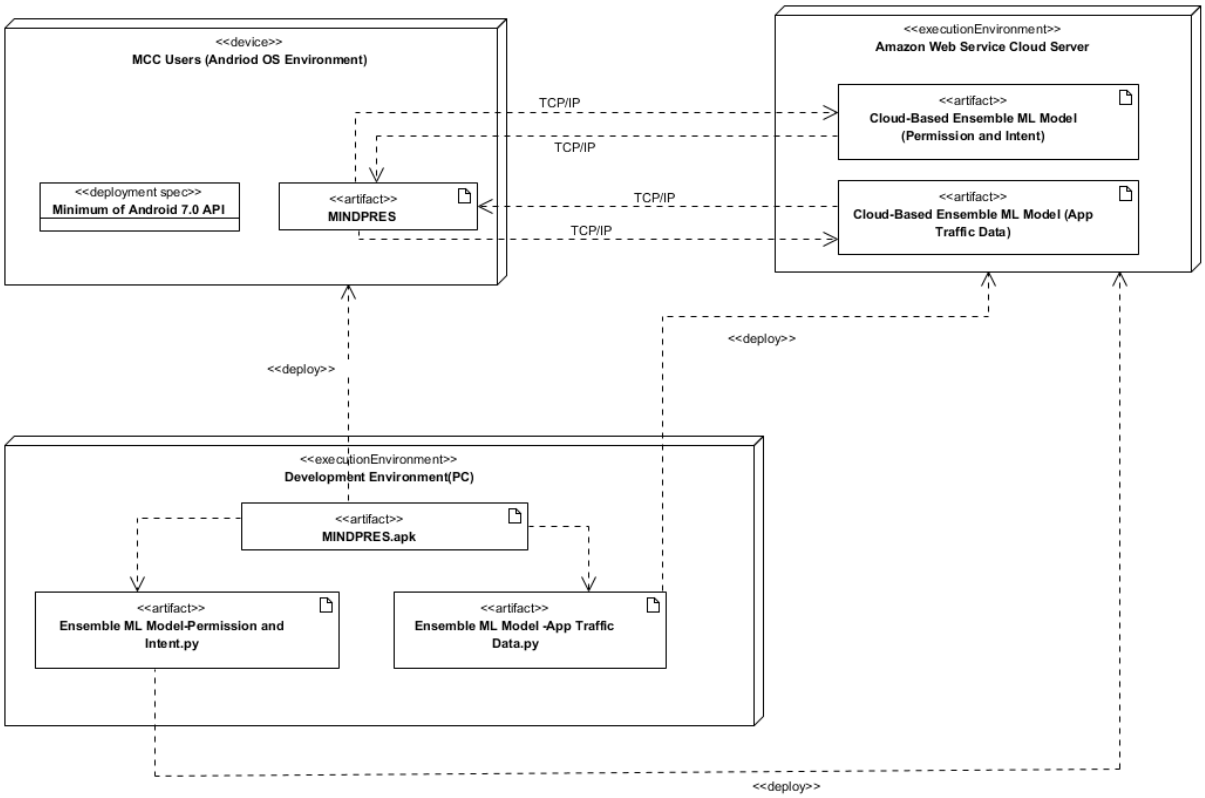

Figure 5. Deployment Diagram of the Prototype System

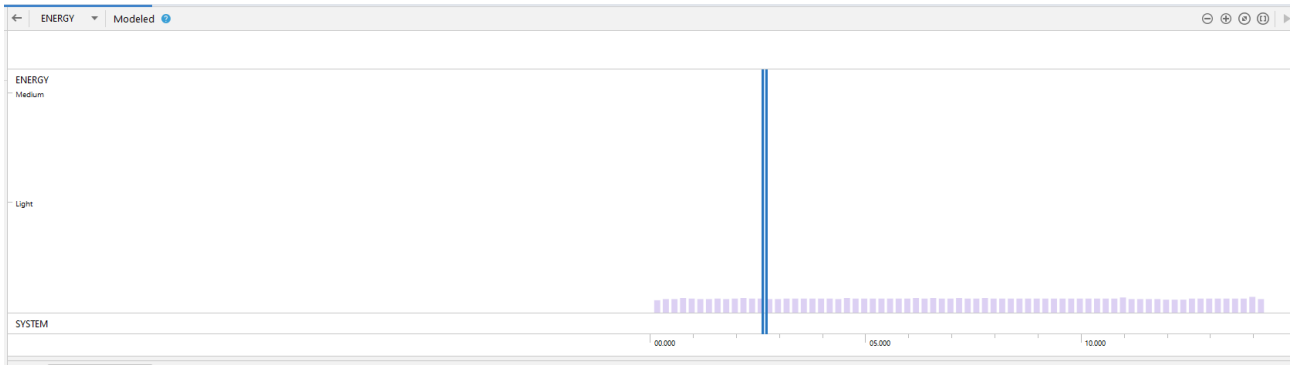

Figure 6. Energy Profiler Estimation Result of the Prototype System

APPENDIX C (SOURCE CODES)

```
package com.mindpres.remote_capture.adapters;
import android.content.Context;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageManager;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import com.mindpres.remote_capture.LocalDb.SQLDB2;
import com.mindpres.remote_capture.R;
import androidx.recyclerview.widget.RecyclerView;
import java.text.DecimalFormat;
import java.util.List;


public class MlAdapter extends RecyclerView.Adapter<MlAdapter.ViewHolder> {

   private List<ApplicationInfo> allApps;
   private Context context;
   private List<Integer> Predictions;
   private List<Double>RiskValues;
   private  ItemClickListener mClickListener;

   // data is passed into the constructor
   public MlAdapter(List<ApplicationInfo> allApps, Context
context,List<Integer>Predictions,List<Double>RiskValues) {
      this.allApps = allApps;
      this.context = context;
      this.RiskValues=RiskValues;
      this.Predictions=Predictions;
   }

   // inflates the row layout from xml when needed
   @Override
   public  ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
      Context context = parent.getContext();
      LayoutInflater inflater = LayoutInflater.from(context);
      View view = inflater.inflate(R.layout.mlrecyclerview, parent, false);
      return new  ViewHolder(view);
   }

   // binds the data to the TextView in each row
   @Override
   public void onBindViewHolder(ViewHolder holder, int position) {
      ApplicationInfo app = allApps.get(position);

      // TextView textView = holder.nameTextView;
      holder.pckimage.setImageDrawable(app.loadIcon(context.getPackageManager()));
      holder.pckname.setText(app.loadLabel(context.getPackageManager()));


      if (Predictions.get(position)==0)
      {
         holder.status1.setVisibility(View.VISIBLE);
         holder.status11.setVisibility(View.INVISIBLE);
```

```
   holder.status2.setVisibility(View.INVISIBLE);

}
else
{
   holder.status2.setVisibility(View.VISIBLE);
   holder.status11.setVisibility(View.INVISIBLE);
   holder.status1.setVisibility(View.INVISIBLE);

}
SQLDB2 sqldata=new SQLDB2(context);
holder.percent1.setVisibility(View.INVISIBLE);
holder.percent2.setVisibility(View.INVISIBLE);
holder.percent3.setVisibility(View.INVISIBLE);
if (RiskValues.get(position)>0.85&&Predictions.get(position)==0)
{

   sqldata.addApp(app.packageName,"1",Integer.toString(app.uid));
   DecimalFormat df=new DecimalFormat("0.00");
   holder.percent3.setProgress(78);
   holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
   holder.percent3.setVisibility(View.VISIBLE);
   return;
}
if (RiskValues.get(position)>0.60&&Predictions.get(position)==1)
{
   sqldata.addApp(app.packageName,"1",Integer.toString(app.uid));
   DecimalFormat df=new DecimalFormat("0.00");
   holder.percent3.setProgress(78);
   holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
   holder.percent3.setVisibility(View.VISIBLE);
   return;
}
if (RiskValues.get(position)>0.65&&Predictions.get(position)==0)
{

   sqldata.addApp(app.packageName,"0",Integer.toString(app.uid));
   DecimalFormat df=new DecimalFormat("0.00");
   holder.percent2.setProgress(50);
   holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
   holder.percent2.setVisibility(View.VISIBLE);
   holder.status2.setVisibility(View.INVISIBLE);
   holder.status11.setVisibility(View.VISIBLE);
   holder.status1.setVisibility(View.INVISIBLE);
   return;
}
if (RiskValues.get(position)<=0.65&&Predictions.get(position)==0)
{

   DecimalFormat df=new DecimalFormat("0.00");
   holder.percent1.setProgress(20);
   holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
   holder.percent1.setVisibility(View.VISIBLE);
   return;
}
if (RiskValues.get(position)>=0.25&&Predictions.get(position)==1)
{

   sqldata.addApp(app.packageName,"0",Integer.toString(app.uid));
```

```
        DecimalFormat df=new DecimalFormat("0.00");
        holder.percent2.setProgress(50);
        holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
        holder.percent2.setVisibility(View.VISIBLE);
        holder.status2.setVisibility(View.INVISIBLE);
        holder.status11.setVisibility(View.VISIBLE);
        holder.status1.setVisibility(View.INVISIBLE);
        return;
    }

    if (RiskValues.get(position)>=0&&Predictions.get(position)==1)
    {
        DecimalFormat df=new DecimalFormat("0.00");
        holder.percent1.setProgress(20);

        holder.percentText.setText("Risk Score: "+df.format(RiskValues.get(position)));
        holder.percent1.setVisibility(View.VISIBLE);

        return;
    }



}

// total number of rows
@Override
public int getItemCount() {
    return allApps.size();
}


// stores and recycles views as they are scrolled off screen
public class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    public ImageView pckimage;
    public TextView pckname;
    public TextView status1;
    public TextView status11;
    public TextView status2;
    public TextView percentText;
    public ProgressBar percent1;
    public ProgressBar percent2;
    public ProgressBar percent3;



    ViewHolder(View itemView) {
        super(itemView);
        pckimage = (ImageView) itemView.findViewById(R.id.imageView);
        pckname = (TextView) itemView.findViewById(R.id.textView);
        status1 = (TextView) itemView.findViewById(R.id.status1);
        status11 = (TextView) itemView.findViewById(R.id.status11);
        status2 = (TextView) itemView.findViewById(R.id.status2);
        percentText = (TextView) itemView.findViewById(R.id.textpercent);
        percent1=(ProgressBar)itemView.findViewById(R.id.progressBar1);
        percent2=(ProgressBar)itemView.findViewById(R.id.progressBar2);
        percent3=(ProgressBar)itemView.findViewById(R.id.progressBar3);
```

```
        itemView.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        if (mClickListener != null) {
            try {
                mClickListener.onItemClick(view, getAdapterPosition());
            } catch (PackageManager.NameNotFoundException e) {
                e.printStackTrace();
            }
        }
    }

  }
  public void setClickListener(ItemClickListener itemClickListener) {
    this.mClickListener = itemClickListener;
  }

  // parent activity will implement this method to respond to click events
  public interface ItemClickListener {
    void onItemClick(View view, int position) throws PackageManager.NameNotFoundException;
  }


}
// convenience method for getting data at click position




import java.util.ArrayList;
import java.util.List;

 class AppRiskModel{

      String permission;
      double riskValue;
  public AppRiskModel(String permission,double riskValue)
      {
        this.permission=permission;
        this.riskValue=riskValue;
      }

 }


public class RiskData {

  List<AppRiskModel> permission=new ArrayList<AppRiskModel>();
  public double AppRiskValue=0;
  public RiskData(String[]data)
  {

    permission.add(new AppRiskModel("android.permission.WRITE_EXTERNAL_STORAGE",0.5769));
    permission.add(new AppRiskModel("android.permission.READ_PHONE_STATE",0.8460));
    permission.add(new AppRiskModel("android.permission.ACCESS_COARSE_LOCATION",0.7154));
    permission.add(new AppRiskModel("android.permission.ACCESS_FINE_LOCATION",0.4982));
```

```
    permission.add(new AppRiskModel("android.permission.GET_TASKS",0.6063));
    permission.add(new AppRiskModel("android.permission.READ_EXTERNAL_STORAGE",0.4816));
    permission.add(new AppRiskModel("android.permission.SYSTEM_ALERT_WINDOW",0.2557));
    permission.add(new AppRiskModel("android.permission.READ_LOGS",0.4991));
    permission.add(new AppRiskModel("android.permission.MOUNT_UNMOUNT_FILESYSTEMS",0.5091));
    permission.add(new AppRiskModel("android.permission.CAMERA",0.3986));
    permission.add(new AppRiskModel("android.permission.RECORD_AUDIO",0.1846));
    permission.add(new AppRiskModel("android.permission.GET_ACCOUNTS",0.3375));
    permission.add(new AppRiskModel("android.permission.CALL_PHONE",0.2115));
    permission.add(new AppRiskModel("android.permission.WRITE_SETTINGS",0.2635));
    permission.add(new AppRiskModel("android.permission.SEND_SMS",0.3707));

    //create arrays of element
    List<Integer>allPs=new ArrayList<Integer>();


    //Initialize and create array with 15 elements
    for(int x=0; x<15; x++){allPs.add(0);}


    //Then check if the app make use of permission and change the value to 1
   for(String appData:data)
   {
     if(permission.get(0).permission.equals(appData)){ allPs.set(0,1);continue;}
     if(permission.get(1).permission.equals(appData)){ allPs.set(1,1);continue;}
     if(permission.get(2).permission.equals(appData)){ allPs.set(2,1);continue;}
     if(permission.get(3).permission.equals(appData)){ allPs.set(3,1);continue;}
     if(permission.get(4).permission.equals(appData)){ allPs.set(4,1);continue;}
     if(permission.get(5).permission.equals(appData)){ allPs.set(5,1);continue;}
     if(permission.get(6).permission.equals(appData)){ allPs.set(6,1);continue;}
     if(permission.get(7).permission.equals(appData)){ allPs.set(7,1);continue;}
     if(permission.get(8).permission.equals(appData)){ allPs.set(8,1);continue;}
     if(permission.get(9).permission.equals(appData)){ allPs.set(9,1);continue;}
     if(permission.get(10).permission.equals(appData)){ allPs.set(10,1);continue;}
     if(permission.get(11).permission.equals(appData)){ allPs.set(11,1);continue;}
     if(permission.get(12).permission.equals(appData)){ allPs.set(12,1);continue;}
     if(permission.get(13).permission.equals(appData)){ allPs.set(13,1);continue;}
     if(permission.get(14).permission.equals(appData)){ allPs.set(14,1);continue;}
   }
   double TotalPermissinRiskValue=0;
   int Total=0;
   for(int x=0; x<15; x++)
   {
     if(allPs.get(x)==0){continue;}
     Total++;
     TotalPermissinRiskValue+=permission.get(x).riskValue;

   }
   if(Total==0){return;}
    AppRiskValue=TotalPermissinRiskValue/Total;

  }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Collections;

namespace MINDPRES_RiskModel
{
    public class App_Risk_Classifier
    {
      public void ComputeAppRiskValue()
        {
            double[] A = new double[15] { 0.9147, 0.9652, 0.6820, 0.5953, 0.5017, 0.3342,
0.2947, 0.3057, 0.3057, 0.1970, 0.2018, 0.1414, 0.1881, 0.1615, 0.1736 };
            double[] B = new double[15] { 0.6361, 0.2584, 0.2495, 0.5829, 0.2672, 0.3058,
0.778, 0.185, 0.152, 0.1934, 0.831, 0.1941, 0.762, 0.57, 0.208 };

            using (var reader = new
StreamReader(@"C:\Users\fogwara\source\repos\MINDPRES_RiskModel\DangerousPermissionDataSet.c
sv"))
            {

                while (!reader.EndOfStream)
                {
                    var line = reader.ReadLine();
                    System.Console.WriteLine(line);
                    var values = line.Split(',');

                    float sum = 0;
                    double TotalPermissionRiskValue = 0;
                    double AppRiskValue = 0;
                    string  AppType="";
                    for (int i=0; i<=values.Length-1; i++)
                    {
                        if (i== values.Length - 1) { AppType = values[i].ToString();
continue; }
                        double PermissionRiskValue = 0;
                        double Riskvalue = 0;
                         Riskvalue = A[i] - (A[i] * B[i]);
                        PermissionRiskValue = Math.Sqrt(Riskvalue);
                        TotalPermissionRiskValue = TotalPermissionRiskValue +
(PermissionRiskValue * Convert.ToInt32(values[i].ToString()));
                        sum += Convert.ToInt32(values[i].ToString());


                    }

                    AppRiskValue = TotalPermissionRiskValue  / sum;
                    string RiskType = " ";
                    if (AppRiskValue >= 0.5) { RiskType = "High"; }
                    else if (AppRiskValue >= 0.25) { RiskType = "Medium"; }
                    else { RiskType = "Low"; }
                    Console.WriteLine("The Risk Value  for this App is = " + AppRiskValue);
                    Console.WriteLine("Total No of Dangerous Permission Use  is =  " + sum);
                    Console.WriteLine("The Risk Classification for this App is " +
RiskType);
                    Console.WriteLine("The Actual App category Type is  " +
AppType.ToString());

                }
            }
```

```
        }
    }
}
```

```
#Voting Ensemble Model for Malicous app detection in Python
#This model has the ability to differentiate a benign app from a malicious app
Intrusions)
import pandas as pd
import numpy as np
import time
import sklearn
import matplotlib.pyplot as plt
import seaborn as sn
from   pandas import read_csv

datapath = r"C:\Users\fogwara\Desktop\DangerousPermissionDataSet.csv" # Selected
Features Dataset


mydataset = read_csv(datapath)
array = mydataset.values
X = array[:,0:39]  #The Selected 8 Features of the Esembling Techniques
y = array[:,-1]    #target Class(The Attack Type)


from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree  import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=1)




X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.125, random_state=500) # 0.125 x 0.8 = 0.1
print("Begining of my proposed Voting Classifier Ensemble Model for Malicious app
detection")
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree  import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
```

```
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import time
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, f1_score, log_loss
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
#from sklearn.ensemble import StackingClassifier

clf1 = DecisionTreeClassifier()
clf2 = RandomForestClassifier()
clf3 = AdaBoostClassifier()
clf4 = GaussianNB ()
clf5 = SGDClassifier(loss="hinge", penalty="l2")
clf6 = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=500)
clf7 = KNeighborsClassifier(n_neighbors=5)
clf8 = LinearDiscriminantAnalysis()
clf9 = LogisticRegression(random_state=500, solver='lbfgs', multi_class='ovr')
clf10 = svm.LinearSVC()
#eclf = VotingClassifier(estimators=[('dt', clf1), ('rf', clf2), ('ad',
clf3),('lda', clf6),('knn', clf7)], voting='soft')
#eclf = VotingClassifier(estimators=[('dt', clf1), ('rf', clf2),  ('ad', clf7)],
voting='hard')
eclf = VotingClassifier(estimators=[('dt', clf1), ('rf', clf2),  ('ad', clf7)],
voting='hard')

bgclf=BaggingClassifier(base_estimator=clf2, n_estimators=23, random_state=500,
bootstrap=True)

#seclf = StackingClassifier(estimators=[('dt', clf1), ('rf', clf2), ('ad', clf7)],
final_estimator=RandomForestClassifier())
#seclf = StackingClassifier(estimators=[('dt', clf1), ('rf', clf2), ('ad', clf7)],
final_estimator=RandomForestClassifier())

start_time = time.time()
clf1.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf1-DecisionTreeClassifier: {:.2}".format(elapsed_time))

start_time = time.time()
clf2.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf2-RandomForestClassifier: {:.2}".format(elapsed_time))

start_time = time.time()
clf3.fit(X_train, y_train)
```

```
elapsed_time = time.time() - start_time
print("Training time for clf3:-AdaBoostClassifier {:.2}".format(elapsed_time))

start_time = time.time()
clf4.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf4:-GaussianNB Classifier  {:.2}".format(elapsed_time))

start_time = time.time()
clf5.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf5:-SGDClassifier {:.2}".format(elapsed_time))

start_time = time.time()
clf6.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf6:-MLPClassifier {:.2}".format(elapsed_time))

start_time = time.time()
clf7.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf7:-KNeighborsClassifier {:.2}".format(elapsed_time))

start_time = time.time()
clf8.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf8:-LinearDiscriminantAnalysis
{:.2}".format(elapsed_time))

start_time = time.time()
clf9.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf9:-LogisticRegression {:.2}".format(elapsed_time))

start_time = time.time()
clf10.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for clf10:-Support Vector Machine {:.2}".format(elapsed_time))


start_time = time.time()
eclf.fit(X_train, y_train)
elapsed_time = time.time() - start_time
print("Training time for Ensemble Voting Classifier: {:.2}".format(elapsed_time))


start_time = time.time()
bgclf.fit(X_train, y_train)
elapsed_time = time.time() - start_time
```

```
print("Training time for Ensemble Bagging Classifier: {:.2}".format(elapsed_time))

#start_time = time.time()
#seclf.fit(X_train, y_train)
#elapsed_time = time.time() - start_time
#print("Training time for Ensemble Stacking Classifier:
{:.2}".format(elapsed_time))

start_time = time.time()
clf1_pred = clf1.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf1-DecisionTreeClassifier: {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf1_pred)
#l_loss = log_loss(y_test, clf1_pred)
#f1 = f1_score(y_test, clf1_pred)
print("DecisionTreeClassifier Accuracy is: " + str(acc))
#print("DecisionTreeClassifier Log Loss is: " + str(l_loss))
#print("DecisionTreeClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf1_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf1_pred, labels=[0,1]))

#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf1_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)


start_time = time.time()
clf2_pred = clf2.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf2-RandomForestClassifier: {:.2}".format(elapsed_time))
```

```
acc = accuracy_score(y_test, clf2_pred)
#l_loss = log_loss(y_test, clf2_pred)
#f1 = f1_score(y_test, clf2_pred)
print("RandomForestClassifier Accuracy is: " + str(acc))
#print("RandomForestClassifier Log Loss is: " + str(l_loss))
#print("RandomForestClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf2_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf2_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf2_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
clf3_pred = clf3.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf3-AdaBoostClassifier {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf3_pred)
#l_loss = log_loss(y_test, clf3_pred)
#f1 = f1_score(y_test, clf3_pred)
print("AdaBoostClassifier Accuracy is: " + str(acc))
#print("AdaBoostClassifier Log Loss is: " + str(l_loss))
#print("AdaBoostClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf3_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf3_pred, labels=[0,1]))
```

```
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf3_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
clf4_pred = clf4.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf4-GaussianNB Classifier  {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf4_pred)
#l_loss = log_loss(y_test, clf4_pred)
#f1 = f1_score(y_test, clf4_pred)
print("GaussianNB Classifier  Accuracy is: " + str(acc))
#print("Support Vector Machine Classfier Log Loss is: " + str(l_loss))
#print("Support Vector Machine Classfier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf4_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf4_pred, labels=[0,1]))

#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf4_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
```

```
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)


start_time = time.time()
clf5_pred = clf5.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf5-SGDClassifier {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf5_pred)
#l_loss = log_loss(y_test, clf5_pred)
#f1 = f1_score(y_test, clf5_pred)
print("SGDClassifier Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf5_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf5_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf5_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)


start_time = time.time()
clf6_pred = clf6.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf6-MLPClassifier {:.2}".format(elapsed_time))
```

```
acc = accuracy_score(y_test, clf6_pred)
#l_loss = log_loss(y_test, clf6_pred)
#f1 = f1_score(y_test, clf6_pred)
print("MLPClassifier Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf6_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf6_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf6_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)


start_time = time.time()
clf7_pred = clf7.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf7-KNeighborsClassifier {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf7_pred)
#l_loss = log_loss(y_test, clf7_pred)
#f1 = f1_score(y_test, clf7_pred)
print("KNeighborsClassifier Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf7_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf7_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
```

```
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf7_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
clf8_pred = clf8.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf8-LinearDiscriminantAnalysis
{:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf8_pred)
#l_loss = log_loss(y_test, clf8_pred)
#f1 = f1_score(y_test, clf8_pred)
print("LinearDiscriminantAnalysis Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf8_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf8_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf8_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
```

```
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
clf9_pred = clf9.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf9-LogisticRegression {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf9_pred)
#l_loss = log_loss(y_test, clf9_pred)
#f1 = f1_score(y_test, clf9_pred)
print("LogisticRegression Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf9_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf9_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf9_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
clf10_pred = clf10.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for clf10-Support Vector Machine {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, clf10_pred)
#l_loss = log_loss(y_test, clf10_pred)
#f1 = f1_score(y_test, clf10_pred)
```

```
print("Support Vector Machine Accuracy is: " + str(acc))
#print("SGDClassifier Log Loss is: " + str(l_loss))
#print("SGDClassifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,clf10_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,clf10_pred, labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=clf10_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
eclf_pred = eclf.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for Ensemble Voting Classifier: {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, eclf_pred)
#l_loss = log_loss(y_test, eclf_pred)
#f1 = f1_score(y_test, eclf_pred)
print("Voting Classifier Accuracy is: " + str(acc))
#print("Voting Classifier Log Loss is: " + str(l_loss))
#print("Voting Classifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,eclf_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,eclf_pred,
labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
```

```
TN = 0
FN = 0
preds=eclf_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1

print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

start_time = time.time()
bgclf_pred = bgclf.predict(X_test)
elapsed_time = time.time() - start_time
print("Testing time for Ensemble Bagging Classifier: {:.2}".format(elapsed_time))

acc = accuracy_score(y_test, bgclf_pred)
#l_loss = log_loss(y_test, eclf_pred)
#f1 = f1_score(y_test, eclf_pred)
print("Ensemble Bagging Classifier Accuracy is: " + str(acc))
#print("Voting Classifier Log Loss is: " + str(l_loss))
#print("Voting Classifier F1 Score is: " + str(f1))
from sklearn import metrics
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test,bgclf_pred, labels=[0,1]))
print("Classification Report:")
print(metrics.classification_report(y_test,bgclf_pred,
labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
TP = 0
FP = 0
TN = 0
FN = 0
preds=bgclf_pred
for i in range(len(preds)):
    if y_test[i]==preds[i]==1:
           TP += 1
    if preds[i]==1 and y_test[i]!=preds[i]:
           FP += 1
    if y_test[i]==preds[i]==0:
           TN += 1
    if preds[i]==0 and y_test[i]!=preds[i]:
           FN += 1
```

```
print("TP",TP,"FP", FP, "TN",TN, "FN", FN)


#start_time = time.time()
#seclf_pred = seclf.predict(X_test)
#elapsed_time = time.time() - start_time
#print("Testing time for Ensemble Stacking Classifier: {:.2}".format(elapsed_time))

#acc = accuracy_score(y_test, seclf_pred)
#l_loss = log_loss(y_test, eclf_pred)
#f1 = f1_score(y_test, eclf_pred)
#print("Ensemble Stacking Classifier Accuracy is: " + str(acc))
#print("Voting Classifier Log Loss is: " + str(l_loss))
#print("Voting Classifier F1 Score is: " + str(f1))
#from sklearn import metrics
#print("Confusion Matrix:")
#print(metrics.confusion_matrix(y_test,seclf_pred, labels=[0,1]))
#print("Classification Report:")
#print(metrics.classification_report(y_test,seclf_pred,
labels=[0,1]))
#For Binary Classfication Report of the Confusion Matrix between Intrusion and
Normal Traffic
#TP = 0
#FP = 0
#TN = 0
#FN = 0
#preds=seclf_pred
#for i in range(len(preds)):
  #  if y_test[i]==preds[i]==1:
    #       TP += 1
    #if preds[i]==1 and y_test[i]!=preds[i]:
    #       FP += 1
   # if y_test[i]==preds[i]==0:
    #       TN += 1
    #if preds[i]==0 and y_test[i]!=preds[i]:
       #    FN += 1

# print("TP",TP,"FP", FP, "TN",TN, "FN", FN)

for clf, label in zip([clf1, clf2, clf3,clf4,clf5, clf6, clf7, clf8, clf9, clf10,
eclf,bgclf], ['Decision Tree', 'Random Forest', 'AdaBoost Classifier','GaussianNB
Classifier','SGDClassifier','MLPClassifier','KNeighborsClassifier','LinearDiscrimin
antAnalysis','LogisticRegression','Support Vector Machine', 'Ensemble
Voting','Ensemble Bagging']):
    scores = cross_val_score(clf, X_val,y_val, scoring='accuracy', cv=10)
    #print(scores)
    print("Cross Validation Accuracy: %0.4f  %0.4f  %0.4f (+/- %0.4f) [%s]" %
(scores.mean(),scores.max(), scores.min(), scores.std(), label))
```

```
    #print("Cross Validation Accuracy: " % (scores.mean(), scores.std(), label))
```